UNIVERSIDADE ESTADUAL DE CAMPINAS

Instituto de Matemática, Estatística e
Computação Científica

Paulo Henrique Cunha Gomes

Programação multiobjetivo aplicada ao desenvolvimento de materiais avançados

CAMPINAS
2018

Paulo Henrique Cunha Gomes

Programação multiobjetivo aplicada ao desenvolvimento de materiais avançados

Dissertação apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Matemática Aplicada e Computacional.

Orientador: Washington Alves de Oliveira

Coorientador: Cristiano Torezzan

Este exemplar corresponde à versão final da dissertação defendida pelo aluno Paulo Henrique Cunha Gomes, e orientada pelo Prof. Dr. Washington Alves de Oliveira.

Campinas

2018

Agência(s) de fomento e nº(s) de processo(s): Não se aplica.
ORCID: https://orcid.org/0000-0001-7091-5845

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

G585p

Gomes, Paulo Henrique Cunha, 1970-
Programação multiobjetivo aplicada ao desenvolvimento de materiais avançados / Paulo Henrique Cunha Gomes. – Campinas, SP : [s.n.], 2018.

Orientador: Washington Alves de Oliveira.
Coorientador: Cristiano Torezzan.
Dissertação (mestrado profissional) – Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1. Programação multiobjetivo. 2. Otimização por enxame de partículas. 3. Ligas de titânio – Modelos matemáticos. 4. Programação linear. 5. Otimização matemática. 6. Programação heurística. I. Oliveira, Washington Alves de, 1977-. II. Torezzan, Cristiano, 1976-. III. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Computação Científica. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Multiobjective programming applied to the development of advanced materials
Palavras-chave em inglês:
Multiobjective programming
Particle swarm optimization
Titanium alloys - Mathematical models
Linear programming
Mathematical optimization
Heuristic programming
Área de concentração: Matemática Aplicada e Computacional
Titulação: Mestre em Matemática Aplicada e Computacional
Banca examinadora:
Washington Alves de Oliveira [Orientador]
Priscila Cristina Berbert Rampazzo
Alessandra Cremasco
Data de defesa: 27-08-2018
Programa de Pós-Graduação: Matemática Aplicada e Computacional

Dissertação de Mestrado Profissional defendida em 27 de agosto de 2018 e aprovada pela banca examinadora composta pelos Profs. Drs.

Prof(a). Dr(a). WASHINGTON ALVES DE OLIVEIRA

Prof(a). Dr(a). PRISCILA CRISTINA BERBERT RAMPAZZO

Prof(a). Dr(a). ALESSANDRA CREMASCO

A Ata da Defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria de Pós-Graduação do Instituto de Matemática, Estatística e Computação Científica.

Ao meu Deus, aos meus pais, a minha esposa e ao meu filho. E a todos os amigos e professores que contribuíram para meu aprendizado dedico.

Agradecimentos

Agradeço ao meu Deus pela oportunidade, não poderia deixar de agradecer meus pais Ézio e Gláucia por todos os apoios. Sou grato ao professor Washington Alves de Oliveira por me receber como aluno e criar as fundações deste trabalho e por moldar o ferro das letras. Sinceramente não poderia deixar de agradecer ao professor Cristiano Torezzan pelo trabalho de erguer os pilares desta dissertação. A minha esposa Ana Rita, devo meu respeito e gratidão sempre preocupada com meu bem estar e me tranquilizando. E ao meu querido filho Pedro Henrique sou grato pela compreensão. Agradeço ao Instituto Federal Goiano pelos incentivos, agradeço ao professor Vicente Pereira de Almeida reitor do Instituto Federal Goiano, agradeço ao professor Claudecir Gonçalves pró-reitor do Instituto Federal Goiano e agradeço a professora Valéria Lima e ao diretor de tecnologia da informação Rodrigo Rodrigues Santana. Todos estes colegas do Instituto Federal Goiano contribuíram com assinaturas a favor do mestrado, ora permitindo minha liberação do trabalho quando necessário, ora reconhecendo a importância da capacitação. A gratidão é inesquecível e não se exala com o tempo. Agradeço aos colegas da Faculdade de Ciências Aplicadas da Unicamp. Agradeço ao tempo por amarelar apenas as páginas desta dissertação, deixando minha gratidão intacta sem manchas. Agradeço ao pessoal da secretaria de pós gradução, muito educados e prestativos. Agradeço ao Instituto de Matemática, Estatística e Computação Científica da Unicamp.

“As oportunidades são como as ondas do mar, cabe a nós pegarmos as ondas..., cabe a Deus prover as ondas em tempo de calmaria. Oportunidade nem sempre é aquilo que se parece elevado ou lucrativo. Haviam dois ladrões na cruz, um reconheceu a sua oportunidade....”

Resumo

Neste trabalho abordamos um problema de otimização multiobjetivo para estudar as propriedades de ligas de titânio que serão representadas por modelos matemáticos. A liga de titânio é revestida por meio de processos específicos para que suas propriedades biofuncionais sejam aprimoradas. Propusemos dois modelos matemáticos polinomiais para representar a dureza e a força de adesão da liga em função dos parâmetros de entrada do processo de revestimento. Estes modelos são preditivos, foram desenvolvidos de forma aproximada baseados em dados experimentais fornecidos por Rafieerad et al. (2017) e tiveram o erro da aproximação minimizado pelo método dos mínimos quadrados e pela abordagem de programação linear. Otimizamos estes modelos implementando um algoritmo heurístico para obtermos valores para as propriedades do titânio. Apresentamos neste trabalho uma metodologia para construir os modelos e otimizá-los. Pela metodologia é possível comparar, discutir e apresentar formas de ajuste de coeficientes de modelos além de disponibilizar um algoritmo heurístico para otimização computacional multiobjetivo destes mesmos modelos. A metodologia pode diminuir os altos custos com experimentos em laboratório, pois somente os resultados computacionais mais indicados para a força de adesão e dureza seriam testados e confirmados em laboratório, evitando testes de novas configurações incertas, configurações de parâmetros de entrada do processo que revestiu a liga de titânio. Quanto à implementação computacional, seguimos princípios de engenharia de software para tornar a implementação extensível e maleável, permitindo a adição de novos modelos e testar sua eficácia. Os dados numéricos obtidos a partir da execução da implementação são exibidos por meio de gráficos através da integração com o software Wolfram Mathematica. Os melhores modelos propostos são os que fornecerão simultaneamente os maiores valores para força de adesão e dureza, pois o problema multiobjetivo é de maximização. A metodologia proposta é uma resposta positiva ao cenário ineficiente de desenvolvimento de materiais avançados descrito em Rafieerad et al. (2017). Novos dados experimentais são aproveitados para reajustar os coeficientes dos modelos de maneira iterativa.

Palavras-chave: programação multiobjetivo, otimização por enxame de partículas, modelos preditivos, heurística, problemas quadráticos.

Abstract

In this work we discuss a multiobjective optimization problem to study the properties of titanium alloys that will be represented by mathematical models. The titanium alloy is coated by specific processes so that its biofunctional properties are improved. We proposed two polynomial mathematical models to represent the hardness and bond strength of the alloy as a function of the input parameters of the coating process. These models are predictive, have been developed roughly based on experimental data provided by Rafieerad et al. (2017) and had minimization error minimized by the least squares method and linear programming approach. We optimize these models by implementing a heuristic algorithm to obtain values for the properties of the titanium in question. We present in this work a methodology to construct the models and to optimize them. Through the methodology it is possible to compare, discuss and present ways of adjusting coefficients of models besides providing a heuristic algorithm for multiobjective computational optimization of these same models. The methodology can reduce the high costs with laboratory experiments, since only the most suitable computational results for adhesion strength and hardness would be tested and confirmed in the laboratory, avoiding tests of new uncertain configurations, input parameter configurations of the process that covered the titanium alloy. Regarding computational implementation, we follow software engineering principles to make the implementation extensible and malleable, allowing the addition of new models and testing their effectiveness. The numerical data obtained from the execution of the implementation are displayed by means of graphics through the integration with Wolfram Mathematica software. The best models proposed are those that will simultaneously provide the highest values for adhesion strength and hardness, since the multiobjective problem is of maximization. The proposed methodology is a positive response to the inefficient advanced material development scenario described in Rafieerad et al. (2017). New experimental data are used to readjust the coefficients of the models in an iterative way.

Keywords: multiobjective programming, particle swarm optimization, predictive models, heuristic, quadratic problems.

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

PM Problema Multiobjetivo

PSO Particle Swarm Optimization.

MOPSO Multiobjective Particle Swarm Optimization.

WM Wolfram Mathematica

HV Hardness Vickers

AD Adesion Strength

QM Quadrados Mínimos

PL Programação Linear

GRASP Greedy Randomized Adaptative Search Procedure

MSP Método da Soma Ponderada

PMOL Problema Multiobjetivo Linear

PASA Pareto Archived Simulated Annealing

SA Simulated Annealing

VNS Variable Neighbordhood Search

GEP Genetic Expression Programming

Lista de Algoritmos

Lista de Códigos

Sumário

Introdução

No contexto do desenvolvimento de materiais avançados procura-se desenvolver novas ligas adequando suas propriedades mecânicas às mais diversas aplicações, neste caminho é necessário submeter os materiais a processos de revestimentos para adequá-los à uma aplicabilidade. Em muitos casos esta adequação torna-se mais difícil por ser necessário alterar simultaneamente e de maneira eficiente duas ou mais propriedades mecânicas do material, a dificuldade está em configurar uma série de parâmetros de testes associados a equipamentos complexos utilizados nos tratamentos de revestimento e em otimizar simultaneamente os valores das propriedades desejadas do material. O sentido de material avançado está relacionado ao projeto e programação do material tendo como objetivo sua aplicabilidade e a escolha do método adequado para se obter valores prévios para as propriedades mecânicas desejadas, se possível valores ótimos. Ressaltamos que tais propriedades mecânicas estão em função de parâmetros de entrada dos processos de revestimento.

As Ligas de titânio do tipo β compostas por Zr, Ta, Nb, Sn, Mb tem recebido uma atenção especial devido ao seu potencial como biomaterial em implantes ortopédicos por apresentar uma combinação de propriedades mecânicas como baixa elasticidade, resistência a biocorrosão e características anti-alérgicas. Entretanto devido a baixa estabilidade a longo prazo e o custo de substituição, modificações na superfície da liga ainda são necessárias para melhorar as propriedades biomecânicas dos implantes ortopédicos e prevenir falhas pós cirurgicas. Nos últimos anos, várias pesquisas foram realizadas para diminuir o desgaste e aumentar a resistência a corrosão. Em trabalhos como Rafieerad et al. (2016, 2017) o desgaste é diminuído com o aumento da dureza pelo revestimento, enquanto a corrosão é diminuída com o aumento da força de adesão pelo revestimento. Neste sentido vários processos de revestimento surgiram para melhorar a biofuncionalidade dos implantes de titânio.

Em Rafieerad et al. (2017) é descrito um cenário ineficiente no tratamento das ligas de titânio em aplicações biofuncionais, sendo a configuração dos parâmetros dos processos de revestimento realizada de maneira experimental e com muitas tentativas e erros implicando no gasto de recursos e tempo. Outro ponto descrito neste cenário ineficiente é que metodologias usuais não são capazes de otimizar simultaneamente as propriedades em questão e que os métodos de inteligência artificial não produzem modelos preditivos explícitos para força de adesão e dureza da liga revestida. Já em Rafieerad et al. (2016)

são propostos modelos específicos para a dureza e força de adesão da liga revestida, os modelos preditivos são construídos por programação genética mas entram no cenário ineficiente por apresentar alta sensibilidade aos dados experimentais iniciais e exigir muitos esforços para ajustar corretamente os coeficientes do modelo.

Apesar dos autores em Rafieerad et al. (2017) darem uma resposta positiva ao cenário ineficiente, propondo então que modelos matemáticos preditivos para a força de adesão e dureza devem ser otimizados por uma abordagem multiobjetivo por ser esta abordagem adequada, não analisaram outras formas determinísticas para ajustes de coeficientes de modelos e não organizaram uma metodologia capaz de discutir apresentar e comparar as formas adequadas de ajustes de coeficientes de modelos matemáticos. Uma abordagem por programação linear é capaz de minimizar a função somatório dos desvios absolutos entre os valores experimentais fornecidos e os valores esperados determinados pelos modelos matemáticos e é uma alternativa de ajuste em relação à programação genética. Os autores fizeram os ajustes de coeficientes de cada modelo aplicando heurística por meio do algoritmo enxame de partículas mono-objetivo de dimensão dez, pois cada modelo possui 10 coeficientes a serem ajustados, sendo fornecido para o ajuste apenas 9 pontos experimentais.

Em resposta positiva ao cenário ineficiente, quanto ao projeto e programação das ligas de titânio quando empregadas como biomaterial, apresentamos em nosso trabalho uma metodologia capaz de organizar, comparar e apresentar formas adequadas de ajuste de modelos matemáticos preditivos para força de adesão e dureza e em seguida otimizar estes modelos sob o contexto multiobjetivo aplicando o algoritmo enxame de partículas implementado neste trabalho. Quanto a forma dos modelos matemáticos trabalhamos com polinômios de Taylor de segunda ordem. Neste caminho nos preocupamos com questões computacionais adicionais em como melhorar a convergência do algoritmo responsável por otimizar nossos modelos, apresentamos todos os dados possíveis referentes às etapas do algoritmo e comparamos com resultados semelhantes obtidos computacionalmente através do software Wolfram Mathematica. Em suma temos como resultado aproximações (pois é o que permite uma abordagem heurística de otimização) da fronteira de Pareto para a força de adesão e dureza. Nossa metodologia indica os melhores resultados computacionais para serem verificados em laboratório, caso tenhamos um resultado confirmado, em teoria nossos modelos matemáticos estão bons, caso não , em decorrência desta verificação temos mais dados experimentais que podem ser adicionados aos dados iniciais e de maneira iterativa realizamos um novo ajuste de coeficientes nos modelos. Para escolhermos os melhores modelos matemáticos preditivos para a força de adesão e dureza indicamos os modelos que apresentarem os maiores valores pois nosso problema é de maximização simultânea da dureza e força de adesão, as quais são melhoradas por processos de revestimentos da liga de titânio.

Objetivos

O objetivo da primeira parte deste trabalho é a produção de modelos matemáticos preditivos que são construídos a partir de dados experimentais publicados na literatura. Na segunda parte, utilizando de uma implementação computacional própria, otimizamos os modelos preditivos sob um enfoque multiobjetivo. Para isto, foi aplicado alguns métodos de escalarização de problemas multiobjetivo e a metodologia enxame de partículas multiobjetivo. Consequentemente, as questões computacionais relativas ao desenvolvimento do código, a verificação e análise da eficácia das regras de implementação e a melhoria da convergência do algoritmo foram alvos de preocupação e interesse. O algoritmo enxame de partículas multiobjetivo proposto é uma adaptação de um enxame de partículas mono-objetivo, portanto os detalhes desta adaptação podem ser melhorados e ajustados conforme a aplicação do algoritmo. De forma resumida, este trabalho objetiva abordar questões teóricas em relação ao desenvolvimento de modelos preditivos com base na programação multiobjetivo e nos métodos de escalarização multiobjetivo, além de abordar a otimização dos modelos via métodos computacionais heurísticos. Algumas aplicações em diversas áreas para o algoritmo enxame de partículas multiobjetivo podem ser encontradas nos trabalhos de Leong (2008); Pasandideh et al. (2013); Lin et al. (2015); Rafieerad et al. (2017) e nas referências contidas neles.

Ainda neste trabalho propomos uma metodologia como resposta positiva ao cenário ineficiente referente ao desenvolvimento de materiais avançados.

Organização do texto

O Capítulo 1 apresenta alguns conceitos iniciais relativos a problemas de programação multiobjetivo. Entre eles, a definição de solução ótima no sentido de Pareto e algumas condições de otimalidade que relaciona o problema multiobjetivo à sua versão mono-objetivo escalarizada. Além disso, os conceitos são ilustrados por meio de vários exemplos. Alguns métodos de escalarização que estabelecem uma relação direta entre as soluções ótimas de um problema multiobjetivo com a sua versão mono-objetivo escalarizada são estudados no Capítulo 2, onde também estudamos alguns métodos computacionais de resolução, mais especificamente, os algoritmos baseados no gradiente da função e algoritmos heurísticos bio-inspirados. O Capítulo 3 aborda o estudo das propriedades das ligas de titânio a partir do trabalho de Rafieerad et al. (2017), onde discutimos sobre a construção dos modelos preditivos, os métodos e os cálculos utilizados para obtê-los. Além disso, descrevemos os detalhes de nossa metodologia aplicada à resolução de problemas multiobjetivo experimentais e explicamos alguns detalhes e conceitos sobre o algoritmo heurístico enxame de partículas multiobjetivo que foi implementado. Os vários resultados obtidos a partir dos testes computacionais são apresentados no Capítulo 4, onde explicamos sobre a

arquitetura do código e sobre como foi possível melhorar a convergência do algoritmo. O Capítulo 5 fornece as conclusões gerais do trabalho e alguns temas para pesquisa futura baseados no trabalho atual.

Capítulo 1

Problemas de programação multiobjetivo

Os problemas de programação multiobjetivo (PM) surgem naturalmente em várias áreas do conhecimento, como na engenharia, em finanças, no desenvolvimento de biomateriais, em teoria dos jogos, entre outras. A principal característica destes problemas está na existência de conflitos entre as funções objetivo, sendo que a ideia de ótimo é definida sobre aquelas soluções onde não é possível melhorar um objetivo sem piorar algum outro, o que em geral fornece um conjunto de soluções ótimas conflitantes. Por exemplo, na área de materiais, as ligas de titânio são revestidas com outros metais para que o titânio tenha suas características de biomaterial melhoradas. Em teoria a força de adesão e a dureza das ligas de titânio são conflitantes quando submetidas a processos de revestimento no contexto dos biomateriais. Assim, as soluções de um PM são consideradas ótimas quando são considerados todos os objetivos simultaneamente, ou seja, a composição da solução ótima do ponto de vista multiobjetivo não é necessariamente formada pelas soluções ótimas individualizadas de cada função objetivo, ficando a cargo do tomador de decisão priorizar aquela solução mais adequada entre as várias possíveis. Além disso, devido aos conflitos existentes entre as funções, dificilmente as soluções ótimas individuais de cada objetivo irão se coincidir para formar uma mesma solução ótima do problema multiobjetivo.

De modo geral um PM é formulado com um conjunto de funções objetivo e um conjunto de restrições, as quais são funções reais de variáveis reais que modelam, por exemplo, quantidade de recursos, limitações de tempo, etc. Na sequência, as definições e conceitos preliminares que explicam melhor o conceito de problemas multiobjetivos são apresentadas. Incluindo uma série de exemplos para facilitar a compreensão deste trabalho.

1.1 Formulação matemática

Os problemas de programação multiobjetivo (PM) têm aplicações em diversos campos que levam a um processo de tomada de decisão, como na área financeira, internet, biomedicina, gestão de negócios, teoria dos jogos, engenharia, entre outros. Os trabalhos de Pareto (1896); Guignard (1969); Bector (1973); Cohon (1978); Chankong and Haimes (1983); Romero (1993); Miettinen (1999); Osuna-Gómez et al. (2000); Liang et al. (2001); Santos et al. (2008); Osuna-Gómez et al. (2010) e as referências contidas neles fornecem uma grande quantidade de resultados importantes na área.

Apesar dos avanços, resolver um problema PM ainda não é considerada uma tarefa fácil. Ou seja, determinar o conjunto de soluções, ou parte dele, requer grande consumo de tempo em processos computacionais atrelados a conceitos teóricos. Os aspectos teóricos que fornecem condições de otimalidade para problemas de programação multiobjetivo são muito importantes para caracterizar as soluções e facilitar o desenvolvimento de métodos numéricos de resolução. De forma geral, o problema de programação multiobjetivo estudado pode ser escrito no seguinte formato padrão.

$$\begin{aligned} &\text{Minimizar} && (f_1(x), f_2(x), \ldots, f_m(x))^T \\ &\text{sujeito a} && x \in X \subseteq \mathbb{R}^n, \end{aligned} \tag{1.1}$$

onde $f_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, m$, são as funções objetivo suficientemente suaves e conflitantes. Nesta abordagem é comum dizer que as funções objetivos são minimizadas simultaneamente. O vetor de decisão $x = (x_1, x_2, \ldots, x_n)^T$ pertence a uma região factível $X \subset \mathbb{R}^n$ não vazia. Se $x \in X$, então x é uma *solução factível*. A imagem de um vetor de decisão é conhecido como *vetor objetivo* e consiste do vetor $f(x) = (f_1(x), f_2(x), \ldots, f_m(x))^T \in \mathbb{R}^m$, isto é, $f : X \to \mathbb{R}^m$.

Por enquanto não vamos fixar as restrições que formam a região factível X. No entanto, ela pode ser composta por restrições representadas por funções e por restrições geométricas que não podem ser representadas por funções.

Definição 1 (Espaço de decisão)**.** *O espaço de decisão X ou região factível para um PM pode ser caracterizado pelo subconjunto $X = \{x \in S \mid g_i(x) \leq 0,\ i = 1, \ldots, p\}$, onde $g_i : S \to \mathbb{R}$ representa a i-ésima restrição analítica do problema, e incluímos qualquer tipo de restrição que não pode ser representada por meio de funções no conjunto $S \subseteq R^n$.*

Esta definição da região factível é a mesma apresentada por Chankong and Haimes (1983), o autor inclui na definição da região factível um subconjunto formado por restrições analíticas e outro subconjunto formado por restrições que não podem ser expressas analiticamente para dar uma ideia mais ampla sobre a região factível. Nesta dissertação vamos considerar apenas as restrições analíticas no espaço decisão, ou seja, $X = \{x \in \mathbb{R}^n \mid g_i(x) \leq 0,\ i = 1, \ldots, p\}$.

A imagem da região factível X forma um subconjunto importante no contexto deste trabalho. As definições 2 e 3 que seguem são adaptadas de Chankong and Haimes (1983)

Definição 2 (Espaço objetivo)**.** *O espaço objetivo ou espaço dos critérios para um PM pode ser caracterizado pelo subconjunto* $F = \{f(x) \mid x \in X\}$*, onde* $f(x) = (f_1(x), \ldots, f_m(x))^T$ *é um vetor objetivo e* $F \subseteq \mathbb{R}^m$*.*

Definição 3 (Dominância)**.** *Dizemos que* $x \in X$ *domina* $y \in X$ *se* $f_i(x) \leq f_i(y)$ *para todo* $i = 1, 2, \ldots, m$ *e existe pelo menos um* j *com* $j = 1, \ldots, m$*, tal que* $f_j(x) < f_j(y)$*. De maneira resumida* x *é melhor que* y *em pelo menos um objetivo e* x *não é pior que* y *em nenhum objetivo.*

Em otimização multiobjetivo o conceito de solução ótima, no sentido de Pareto (1896), é compreendido quando comparamos vetores no espaço objetivo. Dado um vetor objetivo $f(x^*)$ fixo em um ponto x^*, o ponto x^* é considerado ótimo se, dada uma variação na vizinhança de x^*, nenhuma das componentes de $f(x^*)$ pode ser melhorada nesta vizinhança sem degradar qualquer outra componente. No restante deste texto vamos chamar de $N(y) \subseteq X$ uma *vizinhança* de $y \in X$.

1.2 Definição de solução ótima

O conceito de solução ótima para um PM é o mesmo conceito desenvolvido por Vilfredo Pareto em 1896. Do ponto de vista de distribuição de recursos e bem estar social de uma população, Pareto afirmou que a população está em situação de equilíbrio (situação ótima) quanto a distribuição de recursos, enquanto uma possível redistribuição destes recursos implicar simultaneamente em alguma melhora para um dos participantes e piora para algum outro participante da população.

Nos dias de hoje este conceito tem sido empregado nos mais diversos campos da ciência onde é necessário encontrar pontos de equilíbrio em sistemas complexos, em situações onde existem conflitos muito fortes entre os participantes destes sistemas. Na sequência vamos definir as soluções ótimas de Pareto, permitindo, a partir destas explanações, uma compreensão melhor do conceito de otimalidade para problemas multiobjetivo.

Definição 4 (Solução ótima de Pareto, Chankong and Haimes (1983))**.** *Uma solução factível* x^* *é dita solução ótima de Pareto, ou solução eficiente ou ainda solução não-dominada para um PM, se não existe outra solução* $x \in X$ *tal que* x *domina* x^**.*

Definição 5 (Solução localmente ótima de Pareto, Chankong and Haimes (1983))**.** *Uma solução factível* x^* *é dita solução localmente ótima para um PM, se a Definição 4 vale apenas na vizinhança* $N(x^*) \subseteq X$ *de* x^**.*

Note aqui, e nas definições que seguem, que a partir da Definição 5 as soluções na Definição 4 são soluções globalmente ótimas. Convém relembrar de um ponto vista teórico mais geral que sem a hipótese de convexidade ou de convexidade generalizada das funções envolvidas no modelo matemático de um PM podemos apenas caracterizar as soluções localmente ótimas.

Definição 6 (Conjunto de soluções ótimas de Pareto)**.** *O conjunto de todas as soluções ótimas de Pareto é denotado por* X^**. Note que* $X^* \subseteq X$*.*

Definição 7 (Vetor não-dominado)**.** *Se* $x^* \subseteq X^*$*, então* $f(x^*)$ *é um vetor objetivo não-dominado.*

Definição 8 (Conjunto de vetores não-dominados)**.** *O conjunto de todas os vetores objetivo não-dominados ou simplesmente curva de Pareto ou fronteira de Pareto é denotado por* F^**. Note que* $F^* \subseteq F$*.*

Além da Definição 4, existem outras noções de soluções ótimas para um PM, a seguir apresentamos outras duas noções de otimalidade.

Definição 9 (Solução ótima de Pareto fraca, Miettinen (1999))**.** *Uma solução factível* x^* *é dita solução ótima de Pareto fraca, ou solução fracamente ótima de Pareto ou solução fracamente eficiente ou ainda solução fracamente não-dominada para um PM, se não existe outra solução* $x \in X$ *tal que* $f(x) < f(x^*)$*.*

Em resumo uma solução factível x^* é fracamente ótima de Pareto se não é possível melhorar todos os objetivos simultaneamente.

Definição 10 (Solução ótima de Pareto própria, Miettinen (1999))**.** *Uma solução factível* x^* *é dita solução ótima de Pareto própria, ou solução propriamente ótima de Pareto ou solução propriamente eficiente ou ainda solução propriamente não-dominada para um PM, se existe um escalar* $M > 0$*, tal que, para todo* $x \in X$ *satisfazendo* $f_i(x) < f_i(x^*)$*, existe pelo menos um* j*, tal que,* $f_j(x) > f_j(x^*)$ *e* $\dfrac{f_i(x^*) - f_i(x)}{f_j(x) - f_j(x^*)} \leq M$*,* $i, j = 1, \ldots, m, i \neq j$*.*

Em resumo uma solução factível x^* é propriamente ótima de Pareto se é uma solução ótima de Pareto e se os quocientes entre o ganho de um objetivo e a perda em relação aos demais objetivos são limitados. Os três conceitos de soluções ótimas apresentados se relacionam da seguinte maneira

$$\text{Soluções próprias} \implies \text{Soluções de Pareto} \implies \text{Soluções fracas.}$$

Exemplo 1 (Gráfico de um modelo biobjetivo de uma variável real)**.** *As Figuras 1.1 e 1.2 ilustram um exemplo de um conjunto de soluções ótimas de Pareto no espaço de decisão e uma fronteira de Pareto no espaço objetivo, respectivamente. A Figura 1.1 ilustra o gráfico*

de duas funções reais de valores reais que são conflitantes no intervalo $[1, 3]$. *Enquanto a Figura 1.2 ilustra a imagem de* $[1, 3]$ *pela função vetorial* $f(x) = (f_1(x), f_2(x))^T$ *no espaço objetivo.*

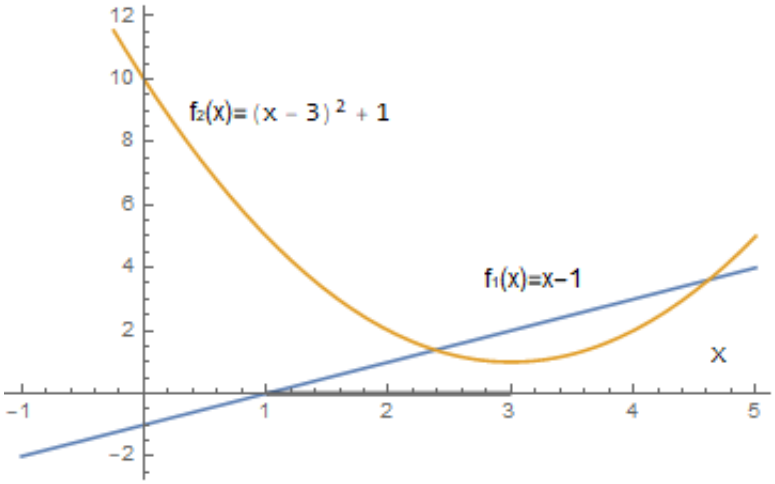

Figura 1.1: Exemplo de um conjunto de soluções ótimas de Pareto $X^* = \{1 \leq x \leq 3\}$. Fonte: adaptado de Chankong and Haimes (1983)

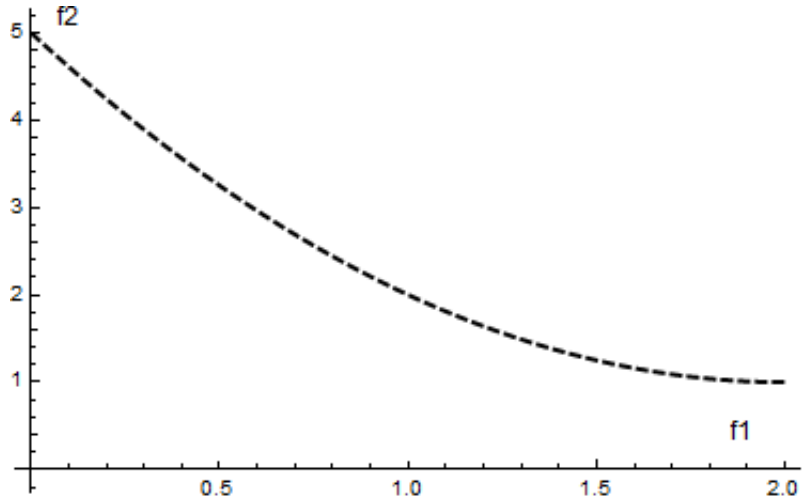

Figura 1.2: Exemplo de uma fronteira de Pareto $F^* = \{f(x^*) \mid 0 \leq f_1(x^*) \leq 2,\ 1 \leq f_2(x^*) \leq 5 \text{ e } 1 \leq x^* \leq 3\}$. Este gráfico é a imagem por $f(x)$ do conjunto X^* da Figura 1.1. Fonte: adaptado de Chankong and Haimes (1983)

Na sequência vamos analisar as soluções de um outro exemplo simples de PM, em que o subconjunto do domínio onde as funções objetivo são conflitantes não é conexo, sendo possível identificar e separar em uma ilustração gráfica a existência de soluções locais e globais. Este exemplo vai ajudar na compreensão da relação de dominância. Relação esta

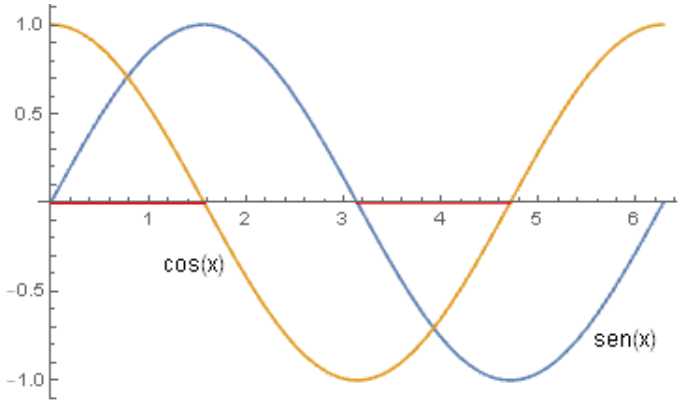

Figura 1.3: Gráfico das funções objetivo, o espaço de decisão $X = \{0 \leq x \leq 2\pi\}$, e o conjunto de soluções ótimas de Pareto $X^* = \{0 < x < \pi/2\} \cup \{\pi < x < 3\pi/2\}$. Fonte própria

fortemente abordada na estratégia de escolha de soluções aproximadas no nosso algoritmo heurístico.

Exemplo 2 (Soluções locais e globais em um modelo biobjetivo de uma variável real)**.** *As Figuras 1.3 e 1.4 ilustram as soluções ótimas de Pareto no espaço de decisão e a fronteira de Pareto no espaço objetivo para o modelo biobjetivo apresentado na Formulação (1.2).*

$$\begin{aligned} &\textit{Minimizar} && f(x) = (cos(x), sen(x))^T \\ &\textit{sujeito a} && 0 \leq x \leq 2\pi. \end{aligned} \tag{1.2}$$

A Figura 1.3 destaca em cor vermelha o conjunto de soluções ótimas de Pareto $X^* = \{0 < x < \pi/2\} \cup \{\pi < x < 3\pi/2\}$*. Note que, dado um ponto fixo* x^* *neste conjunto, as funções* $f_1(x)$ *e* $f_2(x)$ *são conflitantes para pequenas variações de* x^**. Por outro lado, no restante do domínio* X *isso não ocorre. A Figura 1.4 apresenta as duas regiões (Quadrantes 1 e 3 do plano Euclidiano) que fornecem o conjunto de vetores não-dominados. Elas são formadas pela imagem dos intervalos* $0 < x < \pi/2$ *e* $\pi < x < 3\pi/2$ *pelo vetor* $f(x)$*. Para o intervalo* $0 < x < \pi/2$*, enquanto a função* $sen(x)$ *está aumentando no sentido positivo de* x*, a função* $cos(x)$ *está diminuindo. Enquanto o inverso ocorre no intervalo* $\pi < x < 3\pi/2$*.*

De acordo com as Definições 4 e 5 para soluções ótimas e observando a Figura 1.3, podemos ver que X^* *forma o conjunto de soluções localmente ótimas de Pareto. Contudo, ao observar a Figura 1.4, vemos que apenas o subconjunto* $\{\pi < x < 3\pi/2\} \subset X^*$ *contém as soluções globalmente ótimas de Pareto, uma vez que cada solução* $x \in (\pi, 3\pi/2)$ *domina qualquer solução* $y \in (0, \pi/2)$*.*

Como já mencionado, ainda existem várias dificuldades teóricas e computacionais para resolver um problema PM. Determinar o conjunto de soluções ótimas requer grande consumo de tempo em processos computacionais. No entanto, boas condições de otimalidade

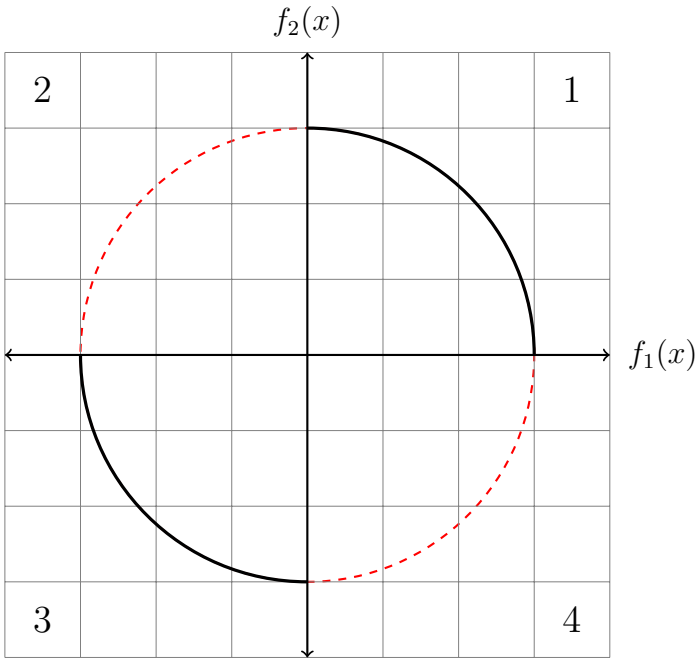

Figura 1.4: Gráfico do espaço objetivo. Fonte própria

que caracterizam as soluções ótimas e têm aspectos interessantes do ponto de vista computacional facilitam o desenvolvimento de métodos numéricos de resolução. A seguir será apresentado uma introdução que fornece as principais condições de otimalidade para problemas de otimização multiobjetivo.

1.3 Condições de otimalidade

Apesar do principal método de resolução abordado nesta dissertação ter base em uma heurística, esta seção fornece uma base teórica inicial sobre algumas condições de otimalidade que são usadas para resolver um problema de otimização e estão de acordo com Chankong and Haimes (1983) e Miettinen (2012).

Definição 11. *O gradiente no ponto x de uma função escalar $f : X \to \mathbb{R}$ suficientemente suave definida em um subconjunto $X \subseteq \mathbb{R}^n$ é calculado utilizando as derivadas parciais e denotado pelo vetor* $\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \ldots, \frac{\partial f(x)}{\partial x_n} \right)^T$.

Teorema 1 (Condição necessária de otimalidade de Fritz John)**.** *Seja x^* uma solução factível Problema (1.1). Se x^* é uma solução localmente ótima, então existem multiplicadores $\lambda \in \mathbb{R}^m$ e $\mu \in \mathbb{R}^p$, com $\lambda \geq 0$, $\mu \geq 0$ e $(\lambda, \mu) \neq (0, 0)$ satisfazendo.*

$$\sum_{i=1}^{m} \lambda_i \nabla f_i(x^*) + \sum_{i=1}^{p} \mu_i \nabla g_i(x^*) = 0 \tag{1.3}$$

$$\mu_i g_i(x^*) = 0, \ \ i = 1, \ldots, p. \tag{1.4}$$

Corolário 2 (Condição necessária de otimalidade de Fritz John para soluções de Pareto fracas)**.** *A condição do Teorema 1 também é necessária se x^* é uma solução fracamente ótima de Pareto.*

A diferença básica entre as condições necessárias de otimalidade de primeira ordem de Fritz John e de Karush-Kuhn-Tucker (KKT) é que no último todas os multiplicadores das funções objetivo (λ) não são iguais a zero, eliminado os casos de modelos degenerados (anormais), onde todas as funções objetivo não desempenham papel algum na solução ótima. No entanto, o preço a pagar é que o conjunto de restrições precisam satisfazer algum tipo de *qualificação de restrições*. Chankong and Haimes (1983) não é uma referência especializada neste tema, porém apresenta os conceitos mais básicos e suficientes para a abordagem tratada nesta dissertação. Vamos supor agora que o conjunto de restrições no Problema (1.1) satisfaz algum tipo de qualificação de restrições para apresentar a próxima condição de otimalidade.

Teorema 3 (Condição necessária de otimalidade de KKT)**.** *Suponha que a solução factível x^* satisfaz as mesmas hipóteses do Teorema 1 e também satisfaz alguma qualificação de restrição. Então o Teorema 1 é válido com a adição que $\lambda \neq 0$.*

Corolário 4 (Condição necessária de otimalidade de KKT para soluções de Pareto fracas)**.** *A condição do Teorema 3 também é necessária se x^* é uma solução fracamente ótima de Pareto.*

1.4 Condições de otimalidade para problemas escalares

Alguns métodos bastante usuais para resolver um PM, chamados de métodos de escalarização, transformam a função objetivo vetorial em uma função objetivo escalar e, lançando mão de alguns resultados teóricos, resolvem o problema escalar (problema mono-objetivo) para obter algumas soluções ótimas de Pareto ou fracamente Pareto.

Algumas condições de otimalidade para um problema mono-objetivo serão apresentadas, porém tratando-se de casos particulares do problema multiobjetivo. Considere o seguinte problema escalar (PE) de otimização.

$$\begin{array}{ll} \text{Minimizar} & f(x) \\ \text{sujeito a} & x \in X = \{x \in \mathbb{R}^n \mid g(x) = (g_1(x), \ldots, g_p(x))^T \leq 0\}, \end{array} \tag{1.5}$$

onde a função objetivo $f : X \to \mathbb{R}$ e as restrições $g_j : X \to \mathbb{R}$, $j = 1, \ldots, p$ são funções reais de valores reais e suficientemente suaves.

Minimizar $f(x)$ significa encontrar uma solução factível x^* tal que $f(x^*) \leq f(x)$ para todo $x \in X$. A solução x^* será um ótimo global de $f(x)$, se a desigualdade $f(x^*) \leq f(x)$

for satisfeita para todo $x \in X$ e, será um ótimo local, se a desigualdade $f(x^*) \leq f(x)$ for satisfeita apenas em alguma vizinhança $N(x^*)$ de x^*.

Teorema 5 (Condição suficiente de otimalidade de KKT)**.** *Seja x^* uma solução factível do Problema (1.5) e suponha que as funções f e g_j, $j = 1, \ldots, p$ sejam convexas. Se existe um multiplicador $\mu \in R^p$, $\mu \geq 0$, tal que*

$$\nabla f(x^*) + \sum_{j=1}^{p} \mu_j \nabla g_j(x^*) = 0 \tag{1.6}$$

$$\mu_j g_j(x^*) = 0 \; j = 1, \ldots, p. \tag{1.7}$$

Então x^* é um solução ótima global do Problema (1.5).

Definição 12. *A matriz hessiana no ponto x de uma função escalar $f : X \to \mathbb{R}$ suficientemente suave definida em um subconjunto $X \subseteq \mathbb{R}^n$ é calculado utilizando as derivadas parciais segundas e denotado pela matriz $\nabla^2 f(x) = \left(\dfrac{\partial^2 f(x)}{\partial x_i \partial x_j} \right)$, $i, j = 1, \ldots, n$.*

Uma matriz $m \times n$ para $m \leq n$ é dita ter posto $r \leq m$ se esta matriz tiver r linhas linearmente independentes. Uma matriz real quadrada $n \times n$ é simétrica se ela é idêntica a sua a sua transposta. Uma matriz M real simétrica $n \times n$ é semidefinida positiva se para todo $y \in \mathbb{R}^n$ obtemos $y^T My \geq 0$, e M é definida positiva se para todo $y \in \mathbb{R}^n$ obtemos $y^T My > 0$. Além disso, M é semidefinida negativa ou definida negativa se para todo $y \in \mathbb{R}^n$ obtemos $y^T My \leq 0$ e $y^T My < 0$, respectivamente. Por outro lado, se exite $y \in \mathbb{R}^n$ e $z \in \mathbb{R}^n$, tal que, $y^T My < 0$ e $z^T Mz > 0$, então M é indefinida.

Problemas sem restrições

Condição necessária de otimalidade: se x^* é um mínimo local do problema $\min_{x \in R^n}\{f(x)\}$, então $\nabla f(x^*) = 0$, isto é, $\dfrac{\partial f(x^*)}{\partial x_i} = 0$ para todo $i = 1, \ldots, n$ e a matriz hessiana $\nabla^2 f(x^*)$ é semidefinida positiva.

Condição suficiente de otimalidade: se no ponto x^* temos $\nabla f(x^*) = 0$ e a matriz hessiana $\nabla^2 f(x^*)$ é definida positiva, então x^* é um mínimo local do problema $\min_{x \in R^n}\{f(x)\}$.

Problemas com restrições de igualdade

Definição 13 (Um tipo de qualificação de restrições para restrições de igualdade)**.** *Seja $h : X \subseteq \mathbb{R}^n \to \mathbb{R}^q$ uma função real de valores reais suficientemente suave. Então x^* é um ponto regular de $h(x) = 0$, se $h(x^*) = 0$ e os vetores gradientes $\nabla h_1(x^*), \ldots, \nabla h_q(x^*)$ no ponto x^* são linearmente independentes.*

Condição suficiente de otimalidade: se x^* é um ponto regular de $h(x) = 0$ e existe $\lambda = (\gamma_1, \ldots, \gamma_q)^T$, tal que,

$$\nabla f(x^*) + \sum_{k=1}^{q} \gamma_k \nabla h_k(x^*) = 0,$$

$$\nabla^2 f(x^*) + \sum_{k=1}^{q} \gamma_k \nabla^2 h_k(x^*) \text{ é uma matriz positiva definida, isto é, para todo } y \neq 0,$$

$$y^T \left(\nabla^2 f(x^*) + \sum_{k=1}^{q} \gamma_k \nabla^2 h_k(x^*) \right) y > 0.$$

Então x^* é um mínimo local do problema $\min_{x \in R^n}\{f(x) \mid h(x) = 0\}$.

Problemas com restrições de desigualdade e igualdade

Definição 14 (Um tipo de qualificação de restrições para restrições de desigualdade e igualdade)**.** *Sejam $g : X \subseteq \mathbb{R}^n \to \mathbb{R}^p$ e $h : X \subseteq \mathbb{R}^n \to \mathbb{R}^q$ funções reais de valores reais suficientemente suaves. Então x^* é um ponto regular de $g(x) \leq 0$ e $h(x) = 0$, se $g(x^*) \leq 0$, $h(x^*) = 0$ e os vetores gradientes $\nabla g_1(x^*), \ldots, \nabla g_p(x^*), \nabla h_1(x^*), \ldots, \nabla h_q(x^*)$ no ponto x^* são linearmente independentes para todo j tal que $g_j(x^*) = 0$, $j = 1, \ldots, p$.*

Condição necessária de otimalidade: se x^* é um ponto regular de $g(x) \leq 0$ e $h(x) = 0$ e um mínimo local do problema $\min_{x \in R^n}\{f(x) \mid g(x) \leq 0,\ h(x) = 0\}$, então existem multiplicadores $\mu \in R^p$ e $\gamma \in R^q$ com $\mu \geq 0$, tal que,

$$\nabla f(x^*) + \sum_{j=1}^{p} \mu_j \nabla g_j(x^*) + \sum_{k=1}^{q} \gamma_k^* \nabla h_k(x^*) = 0,$$

$$\mu_j g_j(x^*) = 0.$$

Essas condições de otimalidade foram a base para muitos dos algoritmos numéricos existentes. Abaixo apresentamos um exemplo descrito por Oliveira (2011), o qual tem a finalidade de ilustrar as definições de soluções ótimas de Pareto e de fronteira de Pareto. Além disso, este exemplo envolve funções objetivo quadráticas sendo que, em um caso mais geral, é este tipo de função objetivo que usamos para obter os resultados principais nesta pesquisa.

Definimos $B(x,r)$ como uma bola aberta de raio r e centro $x \in \mathbb{R}^n$ e $\partial B(x,r)$ a sua fronteira definida pela distância Euclidiana.

Exemplo 3 (Programação biobjetivo quadrático)**.** *Considere a seguinte formulação matemática.*

$$\begin{array}{ll} \textit{Minimizar} & f(x) = \left(4x_1^2 - x_2^2,\ -(x_1-2)^2 + 4(x_2+1)^2\right) \\ \textit{sujeito a} & x = (x_1, x_2) \in \mathbb{R}^2. \end{array} \tag{1.8}$$

Este é um exemplo de programação quadrática biobjetivo sem restrição, onde as funções objetivo são funções quadráticas com matrizes Hessianas indefinidas que podem ser expressas como

$$\begin{array}{rcccl} f_1(x) & = & 0.5\ x^T A_1 x + b_1^T x + c_1 & = & 4x_1^2 - x_2^2, \\ f_2(x) & = & 0.5\ x^T A_2 x + b_2^T x + c_2 & = & -(x_1-2)^2 + 4(x_2+1)^2, \end{array}$$

em que $c_1 = 0$, $c_2 = 0$, $b_1 = (0,0)^T$, $b_2 = (4,8)^T$, *e as matrizes* A_1 *e* A_2 *são*

$$A_1 = \begin{pmatrix} 8 & 0 \\ 0 & -2 \end{pmatrix} \quad e \quad A_2 = \begin{pmatrix} -2 & 0 \\ 0 & 8 \end{pmatrix}.$$

Verificamos para essas matrizes que se $d \neq 0$ *não podem ocorrer simultaneamente*

$$d^T A_1 d \leq 0 \ e\ d^T A_2 d \leq 0, \ \textit{para todo}\ d \in \partial B(0,1). \tag{1.9}$$

De fato, seja $d = (d_1, d_2)^T \neq (0,0)^T$. *Se* $d^T A_1 d = 8d_1^2 - 2d_2^2 \leq 0$ *e* $d^T A_2 d = -2d_1^2 + 8d_2^2 \leq 0$, *então*

$$0 < 6d_1^2 + 6d_2^2 = \left(8d_1^2 - 2d_2^2\right) + \left(8d_2^2 - 2d_1^2\right) \leq 0.$$

O que é uma contradição, portanto (1.9) só ocorre se $d = 0$. *De acordo com o Teorema 3, uma condição necessária para que uma solução* x^* *seja candidata a solução localmente ótima de Pareto é que existam valores* $\lambda_1, \lambda_2 \geq 0$, *não todos nulos, tais que,*

$$\lambda_1 \nabla f_1(x^*) + \lambda_2 \nabla f_2(x^*) = 0. \tag{1.10}$$

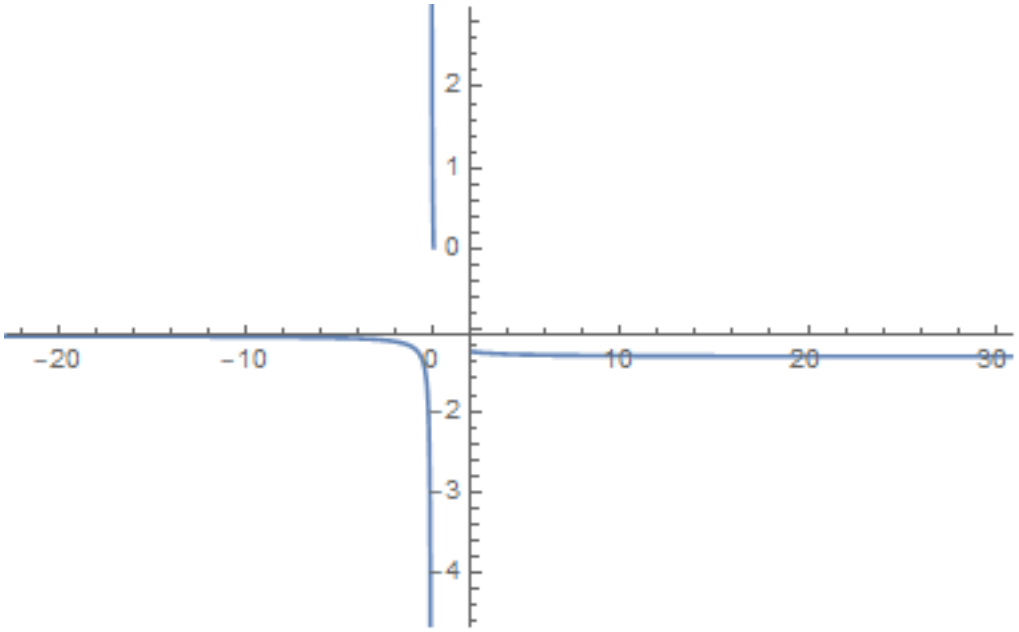

Figura 1.5: Soluções localmente ótimas de Pareto para o Exemplo 3. Fonte: adaptado de Oliveira (2011)

Se $\lambda_2 = 0$ *and* $\lambda_1 > 0$, *então* $x^* = (0,0)^T$. *Note que na direção* $d = (\frac{1}{\sqrt{5}}, \frac{-2}{\sqrt{5}})^T \in \partial B(0,1)$, *para toda nova solução da forma* $x' = x^* + \delta d$, *para o valor real* $\delta \in (0, \frac{4\sqrt{5}}{5})$, *ocorrem simultaneamente* $f_1(x') = f_1(x^*)$ *e* $f_2(x') < f_2(x^*)$. *Portanto, a solução* $x^* = (0,0)^T$ *não é localmente ótima de Pareto. Se* $\lambda_1 = 0$ *e* $\lambda_2 > 0$, *então* $x^* = (2,-1)^T$. *Novamente, note que na direção* $d = (\frac{-2}{\sqrt{5}}, \frac{1}{\sqrt{5}})^T \in \partial B(0,1)$, *para toda nova solução da forma* $x' = x^* + \delta d$, *para o valor real* $\lambda \in (0, 2\sqrt{5})$, *ocorrem simultaneamente* $f_1(x') < f_1(x^*)$ *e* $f_2(x') = f_2(x^*)$. *Portanto, a solução* $x^* = (2,-1)^T$ *não é localmente ótima de Pareto.*

Vamos considerar as possibilidades $\lambda_1 > 0$ *e* $\lambda_2 > 0$, *então (1.10) pode ser reescrito como*

$$\lambda \nabla f_1(x^*) + \nabla f_2(x^*) = 0, \text{ onde } \lambda = \frac{\lambda_1}{\lambda_2} > 0. \tag{1.11}$$

Excluindo $(0,0)^T$ *e* $(2,-1)^T$, *(1.11) também é suficiente para que as demais soluções sejam localmente ótimas de Pareto neste exemplo. De fato, suponha que* x^* *não seja localmente ótimo de Pareto, exite uma direção* $d \in \partial B(0,1)$ *e um número real* $\delta > 0$, *tal que,* $f(x^* + \delta d) \leq f(x^*)$, *com uma desigualdade estrita, é valido. Ou seja, em uma vizinhança de* x^* *existe uma direção de descida* d, *onde* $\nabla f_1(x^*)^T d \leq 0$ *e* $\nabla f_2(x^*)^T d \leq 0$, *com* $\nabla f_1(x^*) \neq 0$ *e* $\nabla f_2(x^*) \neq 0$. *Contudo, para* x^* *e* $\lambda > 0$, *(1.11) é valido, então pelo Teorema de Alternativa de Stiemke (1915),* $\nabla f_1(x^*)^T d < 0$ *e* $\nabla f_2(x^*)^T d \leq 0$ *não tem solução, da mesma forma que* $\nabla f_1(x^*)^T d \leq 0$ *e* $\nabla f_2(x^*)^T d < 0$ *também não tem solução.*

Assim, para que $f(x^* + \delta d) \leq f(x^*)$, *com uma desigualdade estrita e, ao mesmo tempo,*

o Teorema de Alternativa de Stiemke (1915) seja válido, deve ocorrer $\nabla f_1(x^*)^T d = 0$ *e* $\nabla f_2(x^*)^T d = 0$. *Consequentemente deve ocorrer simultaneamente* $d^T A_1 d \leq 0$ *e* $d^T A_2 d < 0$, *ou* $d^T A_1 d < 0$ *e* $d^T A_2 d \leq 0$. *O que é impossível por causa de (1.9). Portanto, as soluções localmente ótimas são todas aquelas que satisfaz (1.11), o que significa* $8\lambda x_1^* - 2(x_1^* - 2) = 0$ *e* $-2\lambda x_2^* + 8(x_2^* + 1) = 0$, *ou ainda*

$$\begin{cases} x_1^* = \frac{2}{-4\lambda+1} \quad , \ 0 < \lambda \neq \frac{1}{4}, \\ \\ x_2^* = \frac{4}{\lambda-4} \quad , \ 0 < \lambda \neq 4. \end{cases} \tag{1.12}$$

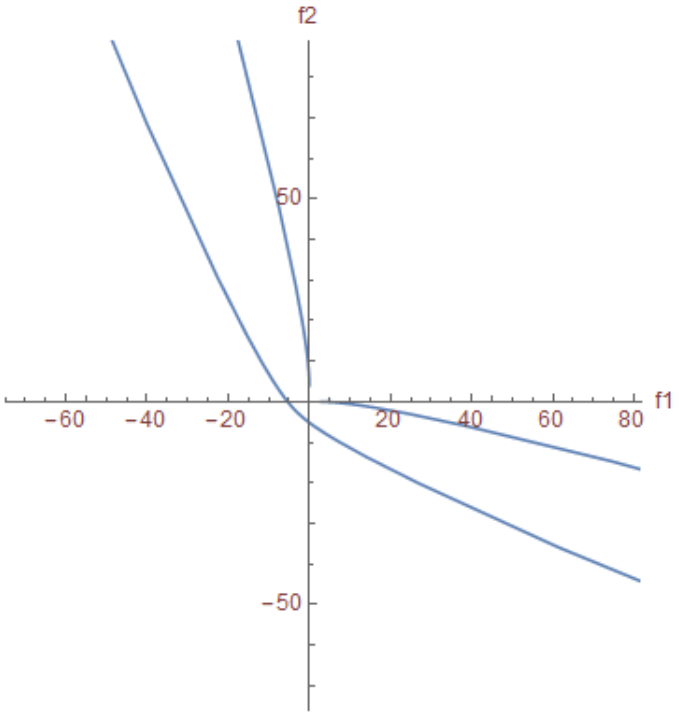

Figura 1.6: Curva de Pareto para as soluções localmente ótimas de Pareto para o Exemplo 3. Fonte: adaptado de Oliveira (2011)

O pontos (x_1^*, x_2^*) *satisfazendo (1.12) são ilustrados na Figura 1.5. Observe que as soluções localmente ótimas de Pareto estão agrupados em três ramos desconectados. Além disso, é possível concluir que o conjunto de soluções ótimas formado pelo desigualdade* $\frac{1}{4} < \lambda < 4$ *são também soluções globalmente ótimas de Pareto. Vamos constatar esta conclusão observando a Figura 1.6. Ela representa a curva de Pareto para este exemplo, ou seja, ela fornece a imagem dos três ramos da Figura 1.5, o que pode ser obtido plotando as funções* f_1 *e* f_2 *nas coordenadas* $(f_1(x_1^*, x_2^*), f_2(x_1^*, x_2^*))^T$, *ou* f_1^* *e* f_2^* *em função de* λ

como em (1.13).

$$\begin{cases} f_1^* = & \frac{16}{(1-4\tau)^2} - \frac{16}{(\tau-4)^2} \\ \\ f_2^* = & \frac{4\tau^2}{(\tau-4)^2} - \left(\frac{2}{1-4\tau} - \frac{4}{\tau-4}\right)^2 \end{cases} \tag{1.13}$$

Observamos que as soluções localmente ótimas fornecida pelo ramo $\frac{1}{4} < \lambda < 4$ *são também soluções globalmente ótimas de Pareto, pois sua imagem fornece os vetores não-dominados na Figura 1.6.*

Neste exemplo o conjunto de soluções globalmente ótimas de Pareto está contido estritamente no conjunto de soluções localmente ótimas de Pareto e a pesar de ser um exemplo simples, investigar as soluções locais ou globais para problemas multiobjetivos não é uma tarefa simples, requer muitas vezes utilizar ferramentas que facilitem a análise, como dispor de métodos computacionais que forneçam as soluções aproximadas.. O modelo matemático principal que é investigado nesta pesquisa também é um modelo quadrático.

1.5 Exemplo de abordagem multiobjetivo em projetos financeiros

A otimização multiobjetivo é muito usada na análise e seleção de projetos de longo prazo de maturação conforme exposto por Abensur (2012). As decisões de investimentos são tomadas com o auxílio de alguns indicadores, como o VPL (Valor Presente Líquido), TIR (Taxa Interna de Retorno), IL (Índice de Lucratividade), PB (Pay Back), PBD (Pay Back Descontado) e MTIR (Taxa Interna de Retorno Modificada). Muitas vezes as decisões são tomadas utilizando funções mono-objetivo e em relação a projetos de investimento excludentes. No entanto, é mais apropriado considerar múltiplos objetivos e múltiplos critérios nas decisões financeiras. É importante ressaltar que os indicadores (atributos) e os pesos escolhidos na especificação de uma possível função objetivo é uma escolha do tomador de decisão. São exemplos na tomada de decisão financeira as políticas que procuram simultaneamente maximizar o retorno e minimizar o risco. Conforme citado no trabalho de Abensur (2012), a tomada de decisão com relação ao orçamento de capitais e análise de investimento é baseada no Índice de Lucratividade no *Pay Back* Descontado e em indicadores de risco.

Definição de projeto conforme Faro and Lachtermacher (2012) é qualquer proposta de financiamento ou investimento que pode ser caracterizado pela sequência $\{a_0, a_1, a_2, \ldots, a_n\}$, onde a_j, para $j = 1, 2, \ldots, n$, representa o fluxo de caixa líquido (a diferença entre receita positiva e despesa negativa ocorridas no final do período j). Um projeto onde $a_0 < 0$ é dito projeto de investimento e um projeto onde $a_0 > 0$ é dito projeto de financiamento.

Critérios de decisão em projetos financeiros

VPL: $V(i) = \sum_{j=0}^{n} \frac{a_j}{(1+i)^j} > 0$, onde $V(i)$ é a função valor atual do projeto de investimento de fluxo de caixa $\{a_0, a_1, a_2, \ldots, a_n\}$, $a_0 < 0$. Baseado no critério VPL, um projeto é viável se $V(i) > 0$. Segundo a análise financeira encontrada em Faro and Lachtermacher (2012), o tomador de decisão terá a opção de investir no projeto, ou seja, a entidade projeto consumirá uma sequência de desembolsos $\{c_0 = -a_0, c_1, \ldots, c_n\}$, produzindo a sequência de receitas $\{b_0 = 0, b1, \ldots, b_n\}$. Optando o tomador de decisão por investir no projeto, ele poderá aplicar as receitas as quais no período n acumularão um montante $M_b(i) = \sum_{j=0}^{n} b_j(1+i)^{n-j}$. Em contra partida, o tomador de decisão poderá optar por não investir no projeto, e aplicar os desembolsos que seriam destinos ao projeto a uma taxa i, acumulando no período n um montante $M_c(i) = \sum_{j=0}^{n} c_j(1+i)^{n-j}$. Portanto, se $M_c(i) < M_b(i)$ é indicado que ele faça o investimento no projeto, até então levando em conta o VPL como um único critério.

TIR: A Taxa Interna de Retorno i^* de um dado projeto é a taxa que anula sua função valor atual $V(i^*) = \sum_{j=0}^{n} \frac{a_j}{(1+i)^j} = 0$. Algumas considerações sobre este critério é que pode ocorrer que o fluxo de caixa do projeto não tenha taxa de retorno, ou seja, a equação $\sum_{j=0}^{n} \frac{a_j}{(1+i)^j} = 0$ não tenha raiz real, outra desvantagem do critério é que pode ocorrer que haja várias TIR's, ou seja, a equação $\sum_{j=0}^{n} \frac{a_j}{(1+i)^j} = 0$ tenha mais de uma raiz real. Este critério para ser usado deve obedecer algumas condições, como a equação deve ter uma única raiz real e não deve conflitar com o critério do valor atual.

PB: O critério do tempo de recuperação de capital ou período de Pay-Back é baseado no tempo de recuperação de capital investido. Este método determina o número mínimo de períodos necessários para que os retornos do projeto superem os investimentos. Este critério pode ser expresso como $\sum_{j=0}^{T} b_j \geq \sum_{j=0}^{T} c_j$, com $\sum_{j=0}^{T-1} b_j < \sum_{j=0}^{T-1} c_j$. Há algumas desvantagens com relação ao uso deste critério, ao somarmos as receitas e despesas, o valor do dinheiro no tempo não é considerado. Por não haver uma taxa de juros, este critério não leva em conta a ordem das parcelas no tempo.

PBD: O critério do tempo de recuperação de capital descontado ou Pay-Back descontado leva em consideração o valor do dinheiro no tempo. Seja um projeto cujo fluxo de caixa é $A = \{a_1, a_2, \ldots, a_n\}$ com $a_j = b_j - c_j$, $j = 0, 1, \ldots, n$, onde o fluxo de caixa líquido é a diferença entre as receitas e os custos sendo esta diferença representada por a_j. O PBD é também baseado no número de períodos T' onde os retornos do projetos são maiores que os investimentos. O PBD é representado como $\sum_{j=0}^{T'} \frac{b_j}{(1+i)^j} \geq \sum_{j=0}^{T'} \frac{c_j}{(1+i)^j}$, com $\sum_{j=0}^{T'-1} \frac{b_j}{(1+i)^j} < \sum_{j=0}^{T'-1} \frac{c_j}{(1+i)^j}$. O projeto deve ser aceito se o período T' for menor ou igual a um limite de tempo estipulado pelo tomador de decisões. Para usarmos este critério em uma função objetivo, pode-se dividir o PDB pelo tempo de vida econômica do projeto.

Risco: Qualquer critério de risco adotado pelo tomador de decisões e expresso em porcentagem.

IL: Índice de lucratividade. Dado um fluxo de caixa com receitas e investimentos ao longo do tempo, projeta-se as receitas obtidas em cada período no período zero e projeta-se os investimentos ocorridas em cada período no período zero, definindo o IL como $IL = \frac{R(i,n) - IV(i,n)}{IV(i,n)}$, onde $R(i,n)$ é o valor presente das receitas e $IV(i,n)$ é o valor presente dos investimentos. Este critério permite que os projetos com vida útil diferentes possam ser comparados através deste índice. O tomador de decisões ainda pode optar por

multiplicar o índice de lucratividade pelo inverso do número de períodos de cada projeto com vida econômica diferentes. Podendo então o IL ser definido de forma alternativa para refletir a duração do projeto, ou seja, $IL = \frac{1}{n}\frac{R(i,n) - IV(i,n)}{IV(i,n)}$, onde n é número de períodos do projeto. Um projeto A, com n_1 períodos, que tenha uma lucratividade maior que um outro projeto B, com n_2 períodos, pode não ser escolhido pelo tomador de decisão se $n_1 > n_2$. Neste caso o tomador de decisões optou por uma lucratividade mais rápida.

Para escolha de projetos dentre várias possibilidades de escolha, pode-se propor o seguinte modelo matemático multiobjetivo de programação linear inteira.

$$\begin{array}{ll} \text{Maximizar} & \sum_{i=1}^{P} x_i IL_i - \sum_{i=1}^{P} x_i PBD_i - \sum_{i=1}^{P} x_i Risco_i \\ \text{sujeito a} & x_i \in \{0,1\}, \end{array}$$

onde o IL, o PDB e o $Risco$ estarão normalizados em porcentagem. Ainda é possível que o tomador de decisões inclua no modelo matemático restrições de projetos mutuamente excludentes, ou inclua restrições para excluir projetos com VPL inferior a valores previamente estipulados.

Capítulo 2

Métodos de resolução

2.1 Métodos de escalarização

Uma das possibilidades de obtenção de soluções ótimas de Pareto é através das soluções ótimas de um problema escalarizado. Através desta estratégia (escalarização), um problema multiobjetivo é transformado em um problema mono-objetivo (subproblema). Em seguida, algoritmos de otimização são empregados na busca das soluções ótimas de cada subproblema. Na literatura encontramos a teoria adequada que relaciona as soluções do problema multiobjetivo original à cada subproblema, além de encontrarmos boas opções de métodos de escalarização. Para resolução computacional de problemas mono-objetivo com muitas variáveis pode-se empregar o algoritmo de busca do gradiente. Os problemas multiobjetivo também podem ser resolvidos de forma direta pela verificação da dominância entre as soluções heurísticas. Devido a complexidade computacional associada aos métodos escalares, complexidade da função objetivo, complexidade de programação do método, complexidade de uso do algoritmo codificado e capacidade do método gerar várias soluções eficientes (veja as análises de Marler and Arora (2004)) os métodos computacionais heurísticos estão sendo utilizados por pesquisadores de diversas áreas por apresentarem bons resultados aproximados e menor complexidade. Em resumo um problema multiobjetivo pode ser resolvido por escalarização ou pela verificação da dominância entre as soluções.

2.1.1 Método da soma ponderada

Conforme é apresentado por Odu and Charles-Owaba (2013), no método da soma de ponderada (MSP) a função objetivo do subproblema a ser resolvido é expressa da forma $\sum_{i=1}^{m} w_i f_i(x)$, onde cada coeficiente $w_i > 0$ representa um peso atribuído a cada função objetivo de acordo com a sua importância entre as demais. Assim, é natural escolher os pesos de tal forma que $\sum_{i=1}^{m} w_i = 1$. As funções objetivo e os coeficientes podem estar em ordem de magnitude diferentes, sendo sempre importante realizar alguma normalização

para obter respostas mais adequadas. Por este método, ao combinar as funções objetivo em uma única função obtém-se apenas uma única solução eficiente, sendo necessário que o algoritmo seja aplicado várias vezes para obtenção de outras soluções eficientes. Se as funções objetivo forem lineares estamos diante de um problema multiobjetivo linear (PMOL).

Em muitos casos a função objetivo escalarizada a ser minimizada ou maximizada é uma função não linear, neste caso é necessário empregar um método computacional de programação não linear ou um método heurístico para resolver o problema.

Teorema 6. *A solução do Problema (2.2) de soma ponderada é uma solução Pareto fraca.*

Este teorema garante que a solução do problema de soma ponderados é Pareto fraca.

Teorema 7. *Seja $F(x)$ uma função de escalarização, definida como a soma ponderada $F(x) = \sum_{i=1}^{k} w_i f_i(x)$, suponha que $\sum_{i=1}^{k} w_i = 1$ e $w_i > 0$ para $\forall i \in \{1, 2, \ldots, k\}$. Se x^* for uma solução ótima obtida usando este método de escalarização da soma de pesos, então esta solução é uma solução eficiente.*

Este teorema nos afirma que se todos os coeficientes de pesos forem positivos a solução ótima do problema de pesos ponderados é também uma solução eficiente do ponto de vista de Pareto.

Estes teoremas nos asseguram que podemos transformar um problema multiobjetivo definido na Seção 1.1, em um problema mono-objetivo da forma

$$\text{Minimizar} \qquad z = \sum_{i=1}^{k} w_i f_i(x) \tag{2.1}$$

$$\text{Sujeito a} \qquad x \in X, \tag{2.2}$$

em que $w_i > 0$, $\forall i \in \{1, 2, \ldots, n\}$ e $\sum_{i=1}^{k} w_i = 1$ e assegura que as soluções ótimas do problema mono-objetivo são também soluções ótimas conforme Pareto.

Deve-se observar que este método pode gerar todo o conjunto de soluções eficientes somente se o espaço objetivo for convexo. Será aplicado o método através de um exemplo biobjetivo, mas antes fazemos algumas considerações sobre o espaço objetivo.

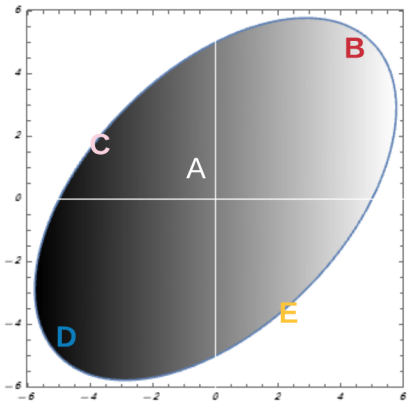

Figura 2.1: Espaço objetivo convexo. Fonte própria

Vemos na Figura 2.1 os vetores A, B, C, D, E, sendo o vetor A o único vetor (da nossa análise) no interior da região objetivo factível e os demais vetores estão na fronteira da região factível (espaço objetivo). O vetor B tem todos os seus objetivos (componentes) maiores que os objetivos de A, já o vetor D tem todos os seus objetivos menores que os de A, já o vetor C tem um dos objetivos menores que os de A e outro maior que A, o mesmo ocorre com o vetor E em relação a A. Podemos observar que todos os vetores que estão no quadrante de B e na fronteira são soluções melhores que A se estivermos pensando em maximização simultânea dos dois objetivos. Também podemos observar que todos os vetores que estão no quadrante de D e na fronteira são soluções melhores que A se estivermos pensando em minimização simultânea dos objetivos. Se caminharmos do ponto A até B estamos melhorando os 2 objetivos simultaneamente, somente caminhando na fronteira é que poderá ocorrer que um objetivo aumente enquanto o outro degrade, ou seja, no interior do espaço objetivo é sempre possível melhorar os dois objetivos. De maneira intuitiva podemos observar que a fronteira de Pareto está na fronteira do espaço objetivo.

Agora será aplicado o método da soma ponderada com a ajuda do exemplo que segue conforme a Figura 2.2. O método consegue gerar toda a fronteira de Pareto se o espaço objetivo for convexo, não podendo ser usado em regiões não convexas. O método consiste em minimizar uma função cujo ótimo também é ótimo conforme Pareto e qual a forma desta função? A função é a função z ponderada, $z = w_1 f_1 + w_2 f_2$ com w_1 e w_2 positivos. Para encontrarmos uma solução eficiente devemos então minimizar a função z restrita a curva $g(f_1, f_2) = 0$. Uma forma de resolver o problema trabalhando no espaço objetivo é aplicar o método dos multiplicadores de Lagrange, $\nabla z(f_1, f_2) = \lambda \nabla g(f_1, f_2)$ com $\nabla g(f_1, f_2) \neq 0$, para mais detalhes sobre o método dos multiplicadores de Lagrange consulte McCallum et al. (2005).

Vejamos agora um exemplo prático descrito em Collette and Siarry (2013).

$$\text{Minimizar} \quad (x^2, (x-2)^2)^T \tag{2.3}$$

$$\text{Sujeito a} \quad x \in [0, 2]. \tag{2.4}$$

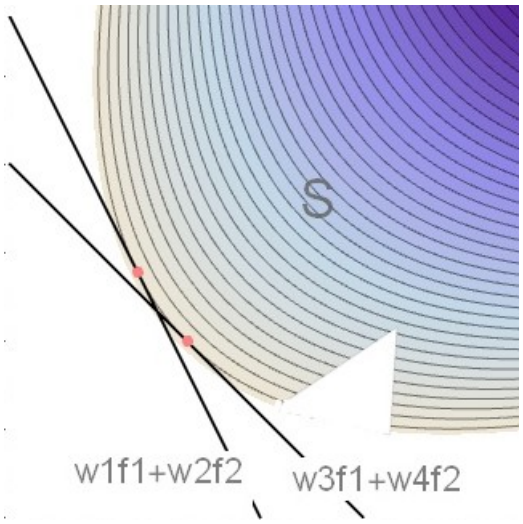

Figura 2.2: Ilustração do método da soma ponderada. Fonte própria

Fazendo $F(x) = w_1x^2 + w_2(x-2)^2$ com $w_1 + w_2 = 1$, podemos reescrever $F(x) = (1-w_2)x^2 + w_2(x-2)^2$ e aplicar as condições de otimalidade $\frac{dF(x)}{dx} = 0$ e $\frac{d^2F(x)}{dx^2} > 0$ para obter $x^* = 2w_2$ e $f_1(x^*) = 4w_2^2$ e $f_2(x^*) = 4w_2^2 - 8w_2 + 4$, agora atribuímos valores a w_2 para gerarmos as soluções de Pareto.

Continuando nossa discussão sobre o método da soma ponderada e apresentando uma solução para o problema 2.3, observe a Figura 2.2 (minimização no espaço objetivo), encontramos o ponto de tangência entre a fronteira do espaço objetivo com as curvas de nível (retas) da função z. Este ponto de tangência é uma solução ótima, podemos então formular o problema da seguinte maneira $z = 0.5f_1 + 0.5f_2$ restrito a $g(f_1, f_2) = f_1^2 + f_2^2 - 2f_1f_2 - 8f_1 - 8f_2 + 16 = 0$ e escolhendo pesos arbitrários $(w_1, w_2) = (0.5, 0.5)$ e fazendo $\nabla z = \lambda \nabla g$ encontramos os valores $(1, 1)$ para (f_1, f_2).

2.1.2 Método épsilon-restrito

Conforme Gandibleux and Ehrgott (2006), este método é baseado na minimização de uma única função objetivo (função primária), considerando as outras funções objetivo como restrições no problema de minimização da função primária. As funções restrições são limitadas por um vetor ε previamente fornecido pelo tomador de decisões. As componentes ε_i do vetor ε são alteradas para gerar toda a fronteira de Pareto. Para cada $k = 1, \ldots, m$

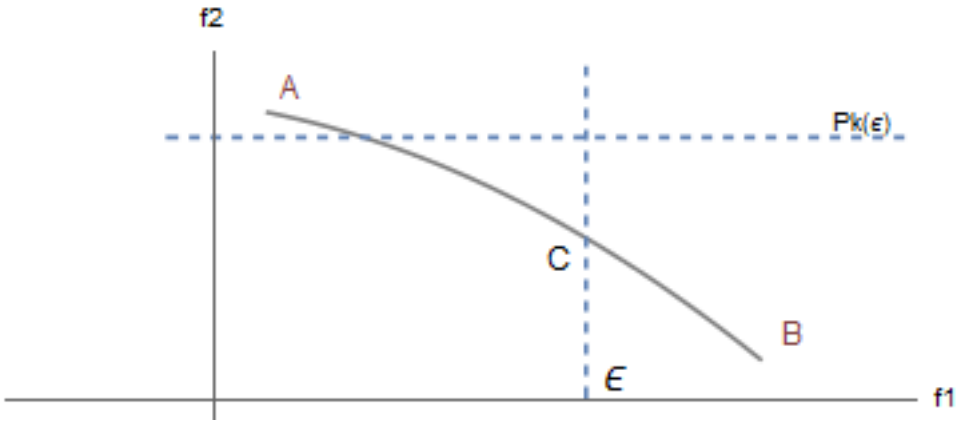

Figura 2.3: Ilustração do método ε-restrito. Fonte: adaptado de Chankong and Haimes (1983)

o método pode ser formulado como segue.

$$\begin{array}{ll} \text{Minimizar} & f_k(x) \\ \text{Sujeito a} & f_i(x) \leq \varepsilon_i \text{ para todo } i = 1, 2, \ldots, m \text{ e } i \neq k \\ & x \in X. \end{array}$$

A vantagem deste método é que ele consegue encontrar soluções onde a fronteira de Pareto não seja convexa, diferentemente do método da soma ponderada.

Vamos aplicar este método em um espaço objetivo com duas dimensões e onde a fronteira de Pareto não seja convexa. A explicação ficará mais clara se observarmos a Figura 2.3, observe que a função objetivo f_k é uma linha horizontal no espaço objetivo para algum valor de ε, e o problema de minimizar $f_k(x)$ se torna um problema de minimizar $f_k(f_i)$ e fazendo $f_i = \varepsilon$ nossa função de minimização no espaço objetivo pode ser escrita como $P_k(\varepsilon)$. Graficamente $P_k(\varepsilon)$ é equivalente a encontrar o menor ponto (menor valor de f_k) da curva AB que fica dentro da região factível de $P_k(\varepsilon)$, assim aumentando ou diminuindo o valor de ε podemos encontrar toda a fronteira de Pareto minimizando $P_k(\varepsilon)$.

A codificação das funções objetivas pode ser bastante complexa se houver muitos objetivos. Este método é empregado em problemas com duas ou três funções objetivas. Conforme Chankong and Haimes (1983) é possível encontrar soluções ótimas de Pareto através de uma variação do método ε-restrito, ao invés de usar desigualdades nas restrições pode-se usar igualdades. Podemos então formular uma variação deste método como segue.

$$\begin{array}{ll} \text{Minimizar} & f_k(x) \\ \text{Sujeito a} & f_i(x) = \varepsilon_i \text{ para todo } i = 1, 2, \ldots, n \text{ e } i \neq k \\ & x \in X. \end{array}$$

Por este método podemos gerar todo o conjunto de Pareto. Ilustraremos a questão a

partir de um exemplo de Chankong and Haimes (1983) que será discutido.

$$\text{Minimizar} \quad (x_1x_2, x_1^2 + x_2^2) \tag{2.5}$$

$$\text{Sujeito a} \quad x_1 + x_2 = 1, x_1 \geq 0, x_2 \geq 0. \tag{2.6}$$

Inicialmente escolhemos f_1 como função primária a ser otimizada e o problema é reescrito como segue.

$$\text{Minimizar} \quad f_1(x) = x_1x_2 \tag{2.7}$$

$$\text{Sujeito a} \quad f_2(x) = x_1^2 + x_2^2 = \varepsilon_2 \tag{2.8}$$

$$g_1(x) = x_1 + x_2 = 1 \tag{2.9}$$

$$g_2(x) = -x_1 \leq 0 \tag{2.10}$$

$$g_3(x) = -x_2 \leq 0. \tag{2.11}$$

Neste exemplo podemos obter uma relação entre as componentes da função a ser minimizada do Problema (2.5). Podemos afirmar que $(x_1 + x_2)^2 = x_1^2 + x_2^2 + 2x_1x_2$ e, pela Equação (2.9), $x_1 + x_2 = 1$, obtemos $x_1^2 + x_2^2 + 2x_1x_2 = 1$, e rescrevendo esta última equação temos $\epsilon_2 + 2f_1(\varepsilon_2) = 1$, agora observemos que $\frac{df_1(\epsilon)}{d\varepsilon_2} = -1/2$ é menor que zero para todo ε_2 (indicação de que f_2 aumenta enquanto f_1 diminui). Basta agora verificar a implicação das restrições na fronteira de Pareto, ou seja, em (f_2, f_1). As condições de não negatividade de x_1 e x_2 implicam que f_2 seja positiva e que f_1 também seja positiva. Considerando o espaço objetivo como o conjunto $F^* = \{(f_2, f_1) \mid 2f_1 + f_2 = 1,\ 0 \leq f_2 \leq 1\}$ e o espaço de decisão como o conjunto $X^* = \{(x_1, x_2) \mid x_1 + x_2 = 1,\ 0 \leq x_1 \leq 1\}$.

2.1.3 Programação por metas

A programação por metas, que é bastante conhecida pelo seu nome em inglês *goal programming*, consiste em aproximar-se ao máximo possível de todos as metas previamente definidas, se desviando o mínimo possível de todas as metas combinadas através de uma função utilitária. Vamos tomar como exemplo o seguinte problema de maneira genérica.

$$\text{Minimizar} \quad (f_1(x), \ldots, f_m(x))^T \tag{2.12}$$

$$\text{Sujeito a} \quad x \in X. \tag{2.13}$$

Fixemos as metas $(f_1^0, f_2^0, \ldots, f_n^0)$ e, em seguida, escolhemos uma alternativa que tenha o menor desvio combinado em relação as metas minimizando a função agregada $U = \sum_{i=1}^{m} P_j|f_i - f_i^0|$, onde P_j são pesos de importância atribuídos aos objetivos pelo tomador de decisões. Conforme Chankong and Haimes (1983) este tipo de problema de decisão, de se aproximar o mais próximo possível de todas as metas é conhecido como programação por metas. Pela minimização da função U obtêm-se uma solução de Pareto.

2.2 Algoritmos em métodos de escalarização

Devido à técnica de escalarização, um problema multiobjetivo é convertido em um problema mono-objetivo, em seguida, estes problemas são abordados computacionalmente quanto a otimização, consequentemente torna se inevitável o estudo de algoritmos numéricos para resolver estes problemas escalarizados e associados a funções matemáticas com muitas variáveis. Neste trabalho estudamos principalmente os algoritmos heurísticos e uma breve revisão de algoritmos determinísticos, em especial o algoritmo do gradiente. Nas próximas seções definiremos conceitos como metaheurística, ilustraremos os tipos de classificações de algoritmos heurísticos e faremos uma revisão sobre os algoritmos busca do gradiente, VNS, PASA, PSO.

2.2.1 Algoritmos com buscas direcionais

Uma classe importante de algoritmo que aparece na literatura é a classe de algoritmos baseados em direções de busca, nesta seção vamos dar uma noção sobre o que seria uma direção de descida.

Se $x \in \mathbb{R}^n$ e $\nabla f(x) \neq 0$ implica que x não é um minimizador local de f em $\mathbb{R}^n$, logo em toda vizinhança de x existe um $z \in \mathbb{R}^n$ tal que $f(z) < f(x)$. Queremos caracterizar direções a partir de x em que é possível achar um ponto $z \in \mathbb{R}^n$ tal que $f(z) < f(x)$. As direções $d \in \mathbb{R}^n$, tais que $\nabla^t f(x)d < 0$, são chamadas direções de descida a partir de x. A existência destas direções sugere uma classe de algoritmos baseados em direções de busca. Para detalhes sobre este tema consulte Friedlander (2012). Na sequência abordamos o algoritmo do gradiente, uma alternativa de buscas direcionais para otimização de problemas escalarizados.

Procedimento busca do gradiente

O procedimento da busca do gradiente é um método particular dos métodos de descida ou subida, respectivamente para minimização e maximização de funções de várias variáveis, este método é conhecido como *speedest descent* ou *descida ingrime* e tem convergência linear. Seja o problema de maximizar uma função $f(x)$ com múltiplas variáveis e sem restrições $x = (x_1, x_2, \ldots, x_n)$, uma condição necessária para que uma solução particular x^* seja ótima quando esta função for diferenciável é que $\frac{\partial f(x^*)}{\partial x_j} = 0$, para $j = 1, 2, \ldots, n$. Se $f(x)$ for concava a condição necessária se torna suficiente também. Se $f(x)$ for uma função linear o problema se reduz a resolver um sistema linear com n equações igualando as derivadas parciais a zero. Mas se $f(x)$ for uma função não linear é provável que não seja possível resolver analiticamente todas as equações cuja as derivadas parciais são igualadas a zero. Então um algoritmo numérico para encontrar máximos ou mínimos de funções não lineares deve ser usado, uma opção é o algoritmo da busca do gradiente conforme descrito por Hillier (2015). Ainda segundo Hillier (2015), adaptações deste algoritmo são

empregadas nos algoritmos de otimização não linear com restrições.

1. Expresse $f(x' + t\nabla f(x'))$ como uma função de t e faça $x_j = x'_j + t(\frac{\partial f(x')}{\partial x_j})$, para todo $j = 1, 2, \ldots, n$, e então substitua estas expressões na função;
2. Use um algoritmo numérico (bissecção, newton) para encontrar máximos e mínimos de funções de uma variável, encontrando $t = t^*$ que maximiza ou minimiza $f(t)$;
3. Reinicie $x' = x' + t\nabla f(x')$;
4. Repita até que $|\frac{\partial f}{\partial x_j}| < \epsilon$ para todo $j = 1, 2, \ldots, n$.

Este algoritmo representa uma ideia chave da programação não linear. Ele identifica a direção do gradiente de $f(x)$, onde a função apresenta maior crescimento a partir de um ponto x' de teste, a direção é mantida enquanto o valor da função aumenta até um novo ponto x onde o gradiente é recalculado neste novo ponto.

No Apêndice A é apresentado a codificação feita para o método do gradiente para um exemplo simples com o intuito de compreender um pouco mais esta abordagem de direções de busca, embora em nosso trabalho o problema de otimização proposto foi resolvido com uma metaheurística.

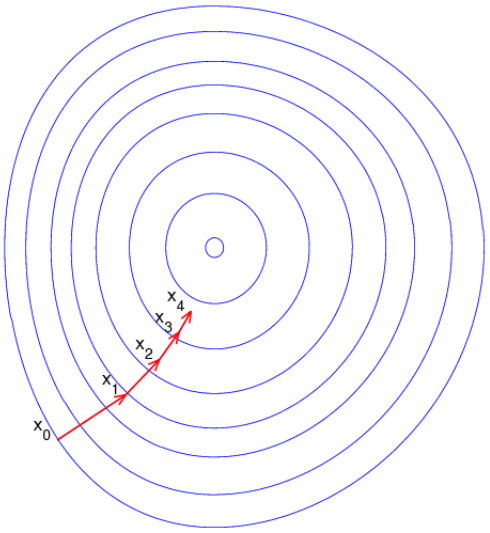

Figura 2.4: Método do gradiente. Fonte Wikipédia

2.2.2 Metaheurísticas para otimização global

Conforme Hillier (2015) uma metaheurística é um método de solução geral que fornece uma estrutura genérica e diretrizes de estratégia para o desenvolvimento de um método heurístico capaz de resolver um determinado tipo de problema. As metaheurísticas tornaram-se uma das técnicas mais importantes no conjunto de ferramentas dos profissionais de Pesquisa Operacional. Uma metaheurística é um tipo geral de solução que orquestra a interação entre um algoritmo ou procedimento de busca local e uma estratégia

de alto nível capaz de escapar de um ótimo local e continuar a busca na região factível por outros ótimos locais, tentando alcançar um ótimo global. Sendo a característica chave de uma boa metaheurística a capacidade de escapar de um ótimo local.

Consideremos o exemplo de programação não linear com múltiplos pontos localmente ótimos: $f(x) = 12x^5 - 975x^4 - 28000x^3 - 345000x^2 - 1.800.000x$ sujeito a $0 \leq x \leq 31$. Usando procedimentos do cálculo ou um software como o Wolfram Mathematica, pode-se verificar que esta função possui os seguintes pontos críticos em $x = 5$, $x = 20$ e $x = 31$. Um algoritmo heurístico capaz de encontrar o ótimo global da função em questão pode ser composto pelo procedimento de *busca do gradiente* aplicado a subintervalos do intervalo $0 \leq x \leq 31$ mais um procedimento recursivo que inicia novamente a busca de pontos ótimos locais em outros intervalos. Poderíamos subdividir o intervalo principal em 5 subintervalos e aplicar o procedimento de busca do gradiente em caso de funções multivariáveis ou aplicarmos o procedimento da bissecção para funções de uma variável, em seguida comparamos os valores dos pontos ótimos locais para encontrarmos o ótimo global. Uma vantagem da metaheurística é encontrar soluções rápidas e onde não é possível aplicar um algoritmo exato, uma desvantagem é que não há garantias que uma solução ótima será encontrada.

2.3 Algoritmos heurísticos de otimização multiobjetivo

Em resumo, algoritmos heurísticos são métodos gerais de otimização dedicados a um problema de difícil resolução, além disso precisam de uma pequena adaptação para serem aplicados em problemas de otimização mais específicos e são baseados em ideias do cotidiano. Podemos citar alguns algoritmos baseados em metaheurísticas como simulated annealing, busca tabu, os genéticos e enxame de partículas, os quais possuem adaptações para problemas multiobjetivo. Conforme Collette and Siarry (2013), há três famílias de algoritmos heurísticos: os determinísticos para pesquisar ótimos locais os quais convergem rapidamente mas não são eficientes para encontrar ótimos globais; os determinísticos para pesquisar ótimos globais; e os estocásticos para pesquisar ótimos globais. Estes últimos são menos eficientes em termos de velocidade, mas encontram pontos ótimos em problemas difíceis de otimização.

Modelos mais gerais de algoritmos de pesquisa são definidos para guiar o desenvolvimento de novos algoritmos capazes de resolverem problemas mais específicos de otimização. Vejamos uma descrição do algoritmo geral 2.1. De forma iterativa um procedimento de geração de soluções candidatas é aplicado à solução atual e um procedimento de substituição da solução atual é realizado. O processo de geração de novos candidatos $C(s)$ e o processo de substituição da solução atual $R(s)$ por alguma solução candidata são processos

que não exigem registro em memória, mas pode-se usar algum histórico de busca armazenado em memória durante o processo de geração de uma lista de soluções candidatas e durante o processo de escolha de uma nova solução.

Algoritmo 2.1 Modelo geral de algoritmos de pesquisa
Input: Solução inicial S_0, $t := 0$;
Repeat
/*Gera soluções candidatas a partir de S_t soluções na vizinhança*/
Generate($C(s_t)$);
/*Seleciona uma solução de $C(S)$ para substituir a solução atual*/
S_{t+1} := Select($C(S_t)$);
$t := t + 1$;
Until critério de parada satisfeito;
Output: Melhor solução.

Definição 15. *Uma função vizinhança V é um mapeamento $V : S \rightarrow 2^S$ que associa cada solução s de S a um conjunto de soluções $V(s) \subset S$. O tipo de vizinhança vai depender do tipo de problema de otimização: problema discreto, otimização em um espaço contínuo, busca em grafo (a vizinhança pode ser considerada o nó adjacente), etc. Uma vizinhança $V(s)$ de uma solução s em um espaço contínuo é a bola com centro em s e raio igual a $\epsilon > 0$, ou seja, $V(s) = \{s' \in R^n \mid \|s' - s\| < \epsilon\}$, usando por exemplo a norma euclidiana $\|s' - s\| = \sqrt{(s'_1 - s_1)^2 + (s'_2 - s_2)^2 + \ldots + (s'_n - s_n)^2}$.*

Para maiores detalhes sobre tipos de vizinhança em algoritmos heurísticos consulte Talbi (2009). São estratégias para a escolha de soluções candidatas na vizinhança de uma solução s: a estratégia do *melhor resultado*, que consiste na busca determinística pelo melhor resultado, podendo demandar um tempo longo na busca de soluções vizinhas; a estratégia do *melhor resultado imediato*, onde basta encontrar um vizinho melhor que a solução atual, implicando em buscas parciais sem ter que avaliar todo um conjunto de soluções vizinhas; e a estratégia da *busca randômica* de vizinhos melhores.

Segundo Talbi (2009), existem algumas classificações para os algoritmos heurísticos, por exemplo, os algoritmos bio-inspirados em processos naturais, como os algoritmos evolucionários, os baseados em sistemas imunológicos da biologia, de colônia de formigas, de abelhas e enxame de partículas. Há também uma classificação baseada no uso de memória ou não. Os algoritmos GRASP, busca local e simulated annealing são exemplos de algoritmos que não usam memória durante as buscas. A busca tabu é exemplo de algoritmo que usa memória. Há também a classificação dos algoritmos em determinísticos versus estocásticos. Nos determinísticos se chega sempre na mesma solução a partir da mesma solução inicial, o que é evitado com os algoritmos estocásticos. Há também uma classificação de busca baseada em populações versus solução única. Já os algoritmos gananciosos (greedy) partem de um conjunto inicialmente vazio de soluções e a cada passo acrescentam novas soluções através de uma heurística local, onde cada elemento

incluído no conjunto solução não deve mais ser substituído por outra solução, não há backtracking (recursividade) em relação a decisões já tomadas.

2.3.1 Algoritmo PASA (*Pareto archived simulated annealing*)

Este método tem uma função que agrega todas as funções objetivo e cataloga as soluções não dominadas. Supõe-se que todas as funções objetivo do problema multiobjetivo sejam positivas. Em consequência desta hipótese podemos usar a seguinte função de agregação $G(\vec{x}) = \sum_{i=1}^{m} ln(f_i(\vec{x}))$. A expressão $\Delta G = G(\vec{x'}) - G(\vec{x}) = \sum_{i=1}^{m} ln(\frac{f_i(\vec{x'})}{f_i(\vec{x})})$ representa a variação média de G entre o ponto $\vec{x}$ e o ponto de teste $\vec{x'}$. Se $\Delta G > 0$ o ponto $\vec{x'}$ piora todas as funções objetivo, se $\Delta G < 0$ o ponto $\vec{x'}$ melhora todas as funções objetivo. Para gerar as soluções não dominadas o método utiliza um arquivo com N soluções. Para obtenção das soluções não dominadas algumas regras são implementadas como segue.

Se $\vec{x'}$ é uma solução dominada por pelo menos alguma solução que está no arquivo, esta não é incluída neste arquivo de soluções não dominadas. Se a solução $\vec{x'}$ domina alguma solução $\vec{y}$ no arquivo, esta substitui a solução dominada no arquivo. Se a solução $\vec{x'}$ não é dominada por soluções do arquivo, ela é incluída neste arquivo e as soluções dominadas por $\vec{x'}$ são removidas do arquivo. Se durante o processo de busca de soluções vizinhas por várias iterações não é encontrado um vizinho melhor, então uma solução ótima é selecionada de maneira randômica do arquivo para que novas buscas se iniciem. São permitidos no máximo 100 iterações sem que se encontre um candidato melhor em relação a solução atual.

Um algoritmo do tipo SA (simulated annealing) não é capaz de retornar um conjunto de Pareto se executado uma única vez. Para preservar as soluções não dominadas obtidas durante o processo de busca, um arquivo de armazenamento é mantido. Veja Collette and Siarry (2013) para um detalhamento sobre este assunto. O PASA é um algoritmo do tipo SA implementa o conceito de vizinhança, aceitação de uma solução nova com alguma probabilidade, dependência da probabilidade no parâmetro temperatura e um modelo de mudança de temperaturas. Em cada iteração, o PASA usa uma população amostral que interage.

Conforme visto em Czyżak and Jaszkiewicz (1998) uma ideia nova usada pelo PASA é o controle de pesos usados nas regras multiobjetivo para influenciar a probabilidade de aceitação de forma a assegurar a dispersão da soluções no conjunto de soluções de Pareto. Quanto maior o peso associado a um objetivo, menor será a probabilidade de aceitar candidatos que causem uma diminuição deste objetivo. Assim, controlando os pesos pode-se aumentar ou diminuir a probabilidade de melhorar os valores de objetivos específicos. Esse cálculo desta probabilidade de aceitação pode ser representado por $P(x, y, T, \Lambda) = \min\{1, \exp \sum_{j=1}^{m} \lambda_j(f_j(x) - f_j(y))/T)\}$ onde x e y são pontos no espaço de busca, T é o parâmetro de temperatura e Λ é o vetor de pesos escolhidos. Uma rotina para o PASA

Algoritmo 2.2 Modelo geral para o PASA

Selecione um conjunto de soluções iniciais $S \subset D$;
Para cada $x \in S$ faça
Atualize o conjunto M de potenciais soluções eficientes com x;
$T := T_0$;
Repita
Para cada $x \in S$ faça
Construa $y \in V(x)$;
Se y não é dominado por x então
Atualize o conjunto M de potenciais soluções de Pareto com y;
$x := y$ com probabilidade $P(x, y, T, \Lambda)$;
Se a condição de mudança de temperatura for satisfeita então
Diminua T;
Fim Repita /*até que a condição de parada seja verificada */

pode ser descrita como no Algoritmo 2.2.

2.3.2 Algoritmo de busca em vizinhança variável (VNS)

O algoritmo VNS para problemas multiobjetivo inicializa a partir de um arquivo contendo soluções não dominadas $D = \{s_1, s_2, s_3, \ldots\}$, o qual seleciona aleatoriamente uma das soluções $s = s_i$ de D e a marca como visitada. Em seguida uma busca numa vizinhança de s é feita para selecionar uma nova solução s'. Após a busca na vizinhança de s, o arquivo D de soluções não dominadas é atualizado com s'. A solução s' é adicionada a D se $s' \notin D$ e não é dominada por nenhuma solução pertencente a D. Após a inclusão de s' em D, todas as soluções de D dominadas por s' são removidas. Então uma nova iteração do algoritmo se inicia escolhendo aleatoriamente uma nova solução s de D não visitada e novas buscas na vizinhança s são realizadas. O algoritmo encerra de acordo com algum critério, como por exemplo o número de iterações máximas atingidas. Este algoritmo multiobjetivo VNS inicia de um repositório já com soluções não dominadas. É crucial para a eficiência deste algoritmo que se defina muito bem a vizinhança a ser pesquisada para que um espaço excessivo de soluções ineficientes não seja pesquisado e o algoritmo se torne ruim devido as buscas ocorrerem longe da fronteira de Pareto. Como exemplo de vizinhança, se o ponto $X_0(x, y, z) = (350, 20, 90)$ pertence a fronteira de Pareto, uma vizinhança de X_0 poderia ser $N(X_0) = \{(351, 20, 90), (350, 21, 90), (350, 20, 91), (351, 21, 90), (351, 21, 91), (350, 21, 91)\}$. Mais detalhes sobre o algoritmo VNS são fornecidos por Gendreau and Potvin (2010) e Arroyo et al. (2011).

Algoritmo 2.3 Modelo do algoritmo PSO mono-objetivo inicial

/*Inicialização randômica das partículas*/
Repeat
Avalie $f(x_i)$
For all Partículas i
Atualize as velocidades
$v_i(t) = v_i(t-1) + \rho_1 x(p_i - x_i(t-1)) + \rho_2 x(p_g - x_i(t-1))$
$x_i(t) = x_i(t-1) + v_i(t)$
if $f(x_i) < f(pbest_i)$ **then** $pbest_i := x_i$
if $f(x_i) < f(gbest)$ **then** $gbest := x_i$
Update (x_i, v_i)
endFor
Until critério de parada satisfeito

2.3.3 Algoritmo PSO multiobjetivo: principais pontos

O algoritmo PSO multiobjetivo (MOPSO) é um PSO, sigla do inglês *Particle Swarm Optimization*, proposto inicialmente por Kennedy and Eberhart (1995). O MOPSO foi discutido por exemplo nos trabalhos de Pasandideh et al. (2013) e Lin et al. (2015). A posição de uma partícula i no espaço d-dimensional de busca é representada por $x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,d})$, enquanto a velocidade desta partícula i é representada por $v_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,d})$. A melhor posição da partícula i durante as buscas é dada por $pbest_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,d})$. A velocidade e a posição das partículas são atualizadas do tempo t para o tempo $t+1$ conforme as equações abaixo.

$$v_{i,d}(t+1) = wv_{i,d}(t) + c_1 r_1(p_{i,d} - x_{i,d}(t) + c_2 r_2(g_i^{repositorio} - x_{i,d}(t)) \tag{2.14}$$

$$x_{i,d}(t+1) = x_{i,d}(t) + v_{i,d}(t+1) \tag{2.15}$$

A atualização da melhor posição da partícula i segue a seguinte regra. Se a nova posição domina a posição $pbest_i$ da partícula, substitua a posição $pbest_i$ pela posição atual $x_i(t+1)$. Se a posição atual é dominada pela posição $pbest_i$, mantenha $pbest_i$. Caso $pbest_i$ não domine a posição atual $x_i(t+1)$, e nem $x_i(t+1)$ domine $pbest_i$, escolha de maneira aleatória a melhor posição individual da partícula i entre $pbest_i$ e $x_i(t+1)$. O Algoritmo 2.3 ilustra uma rotina para o PSO para facilitar o entendimento do MOPSO.

Na atualização da velocidade, o melhor global g_i para uma partícula i sendo atualizada é escolhido aleatoriamente entre as partículas não dominadas do repositório, g_i é o parâmetro *pbest* da solução escolhida aleatoriamente do repositório. Na equação de atualização de velocidade temos $w(t) = (w_{ini} - w_{fim})\frac{(N-t)}{N} + w_{fim}$, onde w é o fator de inércia que direciona as buscas para uma exploração global ou local, e escolhemos $w_{ini} = 0.9$ e $w_{fim} = 0.4$. Os parâmetros c_1 e c_2 são parâmetros de aceleração, em que c_1 e c_2 podem ter ambas o valor 1 ou ambas ter valor 2, ou c_1 e c_2 podem tem soma igual a 4. Podemos observar na literatura que há variações na proposição de algoritmos PSO e MOPSO. En-

contramos uma explanação mais detalhada sobre esses algoritmos na tese de doutorado de Leong (2008).

2.4 Métodos de ajustes de coeficientes em modelos matemáticos

2.4.1 Programação Linear

O problema de ajustes de curvas ou ajuste de coeficientes de modelos consiste em geral em buscar uma lei que rege um fenômeno observável que depende de fatores controláveis. Queremos encontrar uma função para o fenômeno observado, $f(x_1, x_2, \ldots, x_n) = b_i$, onde x_i com $i = 1, 2, \ldots, n$ são variáveis independentes e b_i com $i = 1, 2, \ldots, m$ são valores observados conhecendo os valores correspondentes das variáveis independentes. Geralmente a função f que representa o fenômeno é escolhida entre funções lineares, quadráticas, polinomiais, trigonométricas. De fato encontra-se os coeficientes da função f escolhida como função de aproximação do fenômeno.

2.4.1.1 Modelos de regressão linear

Podemos representar um modelo matemático por meio de uma função linear como segue:

$$b_i = a_1 x_{i1} + a_2 x_{i2} + \ldots + a_n x_{in} + \epsilon_i \quad i = 1, 2, \ldots, m \tag{2.16}$$

Neste caso temos m amostras conhecidas e queremos determinar os coeficientes a_j com $j = 1, 2, \ldots, n$ do modelo.

$A = \{(x_{11}, x_{12}, \ldots, x_{1n}, b_1), (x_{21}, x_{22}, \ldots, x_{2n}, b_2), \ldots, (x_{m1}, x_{m2}, \ldots, x_{mn}, b_m)\}$ sendo ϵ_i ruídos desconhecidos e inevitáveis.

Vamos supor que os ruídos sejam os menores possíveis embora desconhecidos e inevitáveis. Surge então um problema de otimização que consiste em determinar os coeficientes $a_1, a_2, \ldots, a_n$ de modo que os ruídos sejam mínimos em magnitude pois podem ser positivos ou negativos, consequentemente precisamos de uma função para medir o tamanho do ruído $\epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_m)$ na amostra A composta por várias observações. Podemos escrever o problema de otimização dos ruídos da seguinte forma:

$$\text{Minimizar} \qquad f(x_1, x_2, \ldots, x_n, \epsilon_1, \epsilon_2, \ldots, \epsilon_m) = |\epsilon_1| + |\epsilon_2| + \ldots + |\epsilon_m| \tag{2.17}$$

$$\text{Sujeito a} \qquad b_i = a_1 x_{i1} + a_2 x_{i2} + \ldots + a_n x_{in} + \epsilon_i \qquad i = 1, 2, \ldots, m \tag{2.18}$$

O modelo anterior 2.17 é linear por partes e não é diferenciável em $\epsilon_i = 0$ podendo ser reformulado como um modelo de otimização linear redefinindo as variáveis ϵ_i

$$\begin{aligned}
\text{Minimizar} \quad & f(x_1, x_2, \ldots, x_n, \epsilon_1^+, \epsilon_1^-, \epsilon_2^+, \epsilon_2^-, \ldots, \epsilon_m^+, \epsilon_m^-) = \epsilon_1^+ + \epsilon_1^- + \epsilon_2^+ + \epsilon_2^- + \ldots + \epsilon_m^+ + \epsilon_m^- \\
\text{Sujeito a} \quad & b_i = a_1 x_{i1} + a_2 x_{i2} + \ldots + a_n x_{in} + \epsilon_i^+ - \epsilon_i^- \qquad i = 1, 2, \ldots, m \\
& \epsilon_i^+ \geq 0, \epsilon_i^- \geq 0 \qquad i = 1, 2, \ldots, m
\end{aligned}$$

Para uma explicação mais detalhada sobre regressão linear consulte Arenales et al. (2015)

2.4.2 Aproximação de funções por quadrados mínimos

O problema de aproximação de uma função por outra pode ser definido como: Dada uma função f contínua num intervalo $[a, b]$, encontre a melhor aproximação possível de f entre todas as funções de um subespaço W especificado de $C[a, b]$. Vamos ilustrar com exemplos: Encontrar a melhor aproximação de e^x em [0,1] por um polinômio da forma $a_0 + a_1 x + a_2 x^2$ ou encontrar a melhor aproximação de $sen\pi x$ em $[-1, 1]$ por uma função da forma $a_0 + a_1 e^x + a_2 e^{2x} + a_3 e^{3x}$, encontrar a melhor aproximação de x em $[0, 2\pi]$ por uma função da forma $a_0 + a_1 senx + a_2 sen2x + b_1 cosx + b_2 cos2x$. Para o primeiro exemplo W é o subespaço de $C[0, 1]$ gerado pela base $(1, x, x^2)$, no segundo exemplo W é o subespaço de $C[-1, 1]$ gerado pela base $(1, e^x, e^{2x}, e^{3x})$ e para o terceiro exemplo W é o subespaço de $C[0, 2\pi]$ gerado pela base $(1, senx, sen2x, cosx, cos2x)$.

Uma maneira precisa de quantificarmos o erro quando aproximamos uma função $f(x)$ por outra função $g(x)$ no intervalo $[a, b]$ é definirmos o erro no intervalo como erro$=\int_a^b |f(x) - g(x)|dx$. Geometricamente o erro é a área entre as funções. Em suma os matemáticos e cientistas definem o erro como erro$=\int_a^b [f(x) - g(x)]^2 dx$. Logo nossa função $g(x)$ é a função que mais se aproxima de $f(x)$ quando o erro é mínimo. Para uma explicação mais detalhada sobre aproximação de funções por quadrados mínimos consulte Anton and Rorres (2010)

2.4.3 Solução de quadrados mínimos de sistemas lineares

Suponha que o sistema $Ax = b$ seja um sistema linear inconsistente de m equações e n incógnitas onde por hipótese dizemos que a inconsistência foi causada por erros de medição nos coeficientes da matriz A. Devido a inconsistência não é possível encontrar alguma solução exata, mas vamos procurar um vetor x que chegue perto de ser uma solução, vamos encontrar x que minimiza $||b - Ax||$ em relação ao produto interno euclidiano de $\mathbb{R}^m$.

Uma maneira de encontrar alguma solução de mínimos quadrados de $Ax = b$ é calcular a projeção ortogonal $proj_w b$ no espaço coluna W da matriz A e depois resolver a equação $Ax = proj_w b$, entretanto o calculo da projeção pode ser evitado reescrevendo $b - Ax = b - proj_w b$ e então multiplicar ambos os lados da equação por A^T resultando em uma nova

equação $A^T(b - Ax) = A^T(b - proj_w b)$. Como $b - proj_w b$ é ortogonal ao espaço coluna de A esse vetor está no espaço nulo de A^T implicando em $A^T(b - proj_w b) = 0$, ou seja, $A^T(b - Ax) = 0$ que pode ser reescrito como $A^T Ax = A^T b$.

O termo solução de mínimos quadrados decorre do fato: Se $||b - Ax||$ é minimizado $||b - Ax||^2$ também é minimizado. Para uma explicação mais detalhada sobre solução de quadrados mínimos de sistemas lineares consulte Anton and Rorres (2010)

2.4.4 Redes neurais artificiais (RNA) como aproximador universal de funções

Um dos fatos mais impressionantes sobre redes neurais é que elas podem aproximar funções. Isto é, suponhamos que temos uma função complicada $f(x)$, não importa qual seja a função, é garantido que existe uma rede neural de modo que, para cada entrada possível, x, o valor $f(x)$ (ou alguma aproximação) seja calculado pela rede. Este resultado nos diz que as redes neurais têm um tipo de universalidade, não importa qual função queremos aproximar, sabemos que existe uma rede neural que pode calcular o valor próximo da função. Além do mais, pelo teorema da universalidade é válido afirmar que mesmo se restringirmos nossas redes a uma única camada intermediária entre os neurônios de entrada e de saída, uma camada oculta única, é possível aproximar o valor de uma função dada por uma rede neural. Portanto, mesmo arquiteturas de rede muito simples podem ser extremamente poderosas. As redes neurais formadas por neurônios sigmoides $g(u) = \frac{1}{1+e^{-\beta u}}$ e $g(u) = \frac{1-e^{-\beta u}}{1+e^{-\beta u}}$ podem calcular qualquer função. Um tipo de rede neural chamada Perceptron com uma camada oculta pode ser usado como um aproximador universal de função. Uma rede neural artificial é representada por neurônios artificiais, sendo que os principais elementos de um neurônio artificial são: sinais de entrada $\{x_1, x_2, \ldots, x_n\}$, pesos sinápticos $\{w_1, w_2, \ldots, w_n\}$, combinador linear $\sum$, limiar de ativação $\{\theta\}$, potencial de ativação $\{u\}$, função de ativação $g(u)$ e sinal de saída $\{y\}$. O modelo matemático de um neurônio pode ser representado como segue:

$$u = \sum_{i=1}^{n} w_i.x_i - \theta \tag{2.19}$$

$$y = g(u) \tag{2.20}$$

Uma das principais características de uma RNA é sua capacidade de aprendizado por intermédio da aplicação de um método de treinamento, a rede consegue extrair o relacionamento existente entre as variáveis que compõe a aplicação. O processo de treinamento da rede consiste na aplicação de um conjunto de passos ordenados com intuito de ajustar os pesos w e os limiares de seus neurônios. O processo de ajuste também conhecido como algoritmo de aprendizagem visa sincronizar a rede para que suas respostas estejam próximas dos valores desejados. O conjunto de amostras disponíveis sobre o sistema geralmente

é dividido em dois subconjuntos, um subconjunto de treinamento e outro de testes, nas proporções de 60% a 90% para treinamento e 10% a 40% para testes.

Uma prova do teorema da aproximação universal é dada em Cybenko (1989) e em Silva et al. (2010) podemos encontrar uma abordagem inicial sobre redes neurais artificiais.

2.4.5 Programação por expressão genética

A programa genética é uma versão de extensão de algoritmos genéticos é considerada um ramo da computação evolutiva foi inventada por Cramer (1985). A programação genética está de acordo com a teoria da seleção natural de Darwin. A principal diferença entre a programação genética e os algoritmos genéticos esta na sua estrutura individual. Nos algoritmos genéticos os indivíduos são cadeias binárias de comprimento fixo (cromossomas) codificadas linearmente enquanto na programação genética os indivíduos são programas de computador cujo os dados são representados computacionalmente por estruturas de dados chamadas árvores de expressão simbólica ou *parse trees* Gene (2001). Na programação genética os nós da árvore de expressão podem conter funções, variáveis e constantes. As funções na programação genética são compostas por funções básicas da matemática $f = \{+, *, /, -, \sqrt{}, \cos, \sin, \log, ln. \ldots\}$ e os operadores lógicos $\{or, and, not, nor, \ldots\}$. Existem três importantes operadores genéticos que são aplicados nas estruturas de árvores dos indivíduos, respectivamente: reprodução, cruzamento e mutação. Os operadores genéticos operam na árvore modificando e trocando ramos entre as árvores.

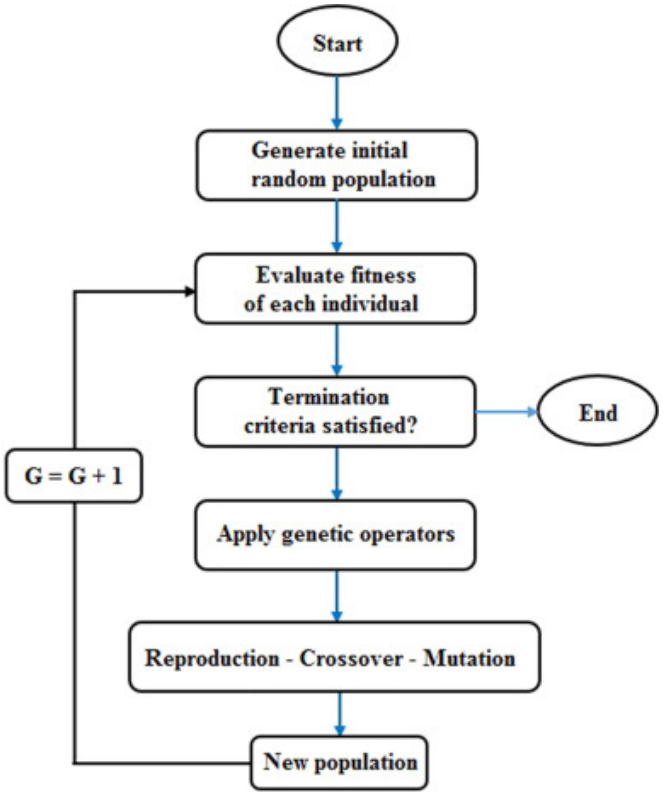

Figura 2.5: O algoritmo GEP. Fonte: Faradonbeh et al. (2016)

Em suma um indivíduo é representado por uma árvore de expressões, operações computacionais de modificação e trocas nos ramos da árvore de expressão são realizadas para se encontrar a função matemática e os coeficientes que melhor se ajustam aos dados observados. O tema cálculo de expressões matemáticas por meio de árvore de expressões já possue alguma décadas e permite que as expressões sejam calculadas sem ambiguidades e sem uso de parenteses.

Capítulo 3

Metodologia própria para construção e otimização multiobjetivo de modelos preditivos.

3.1 Formulação multiobjetivo do problema

De forma geral, o problema de programação multiobjetivo estudado pode ser escrito no seguinte formato padrão.

$$\text{Maximizar} \quad (f_1(x), f_2(x))^T \tag{3.1}$$

$$\text{sujeito a} \quad x \in D, \tag{3.2}$$

onde f_1 e f_2, são funções objetivo suficientemente suaves e conflitantes.

3.2 O ajuste e otimização dos modelos pela metodologia

A partir de dados experimentais, ajustamos modelos matemáticos (funções preditivas) iniciais ao calcular alguns coeficientes para os modelos previamente escolhidos, utilizando duas abordagens: desvio absoluto e mínimos quadrados. Com essas funções, definimos um modelo matemático multiobjetivo e geramos computacionalmente uma aproximação da fronteira de Pareto. Somente os pontos da fronteira de Pareto obtidos como resposta deveriam ser testados em laboratório em novas rodadas de experimentos. Caso não se verifique uma boa precisão dos modelos preditivos, teríamos novos dados experimentais que podem ser incorporados às tabelas de dados experimentais iniciais. Cada novo dado incorporado irá contribuir para uma nova iteração de ajuste de modelos preditivos.

Computacionalmente uma aproximação da fronteira de Pareto é gerada pelo algoritmo

enxame de partículas multiobjetivo. Quanto a quantidade de dados experimentais esta metodologia se aplica onde inicialmente os dados são escassos (caros de se obterem), e se aplica onde os dados já são suficientes (as equações dos modelos não ficam indeterminadas). De forma mais geral esta metodologia se aplica onde há conflitos de objetivos, ou seja, onde a otimização dos objetivos precisa ser tratada neste contexto de conflitos (multiobjetivo).

A Figura 3.1, na parte Modelagem Matemática Computacional (em cor azul), ilustra a descrição em etapas da nossa metodologia. Nela, as etapas que estão contornadas em destaque por cor lilas fazem parte diretamente da metodologia, as etapas são: *1-Minimização de erros*, *2-PL*, *3-QM*, *4-Modelos preditivos que explicam os dados*, *5-Otimizações*, *6-Encontrar soluções adequadas*, e *7-Selecionar as melhores soluções*. A numeração das etapas não representa um ordenamento das etapas. Resumidamente, podemos descrever a metodologia como segue. Recebemos os dados experimentais, calculamos os coeficientes dos modelos pela abordagem Quadrados Mínimos ou Programação Linear, em seguida, otimizamos os modelos executando o algoritmo MOPSO e, finalmente, selecionamos as melhores soluções que são boas aproximações para as *soluções ótimas de Pareto*. Pela metodologia é possível testar outros modelos prontos recebidos diretamente para otimização, como exemplo temos o trabalho de Rafieerad et al. (2016), onde é proposto outros modelos para AD e HV diferentes dos modelos propostos em Rafieerad et al. (2017).

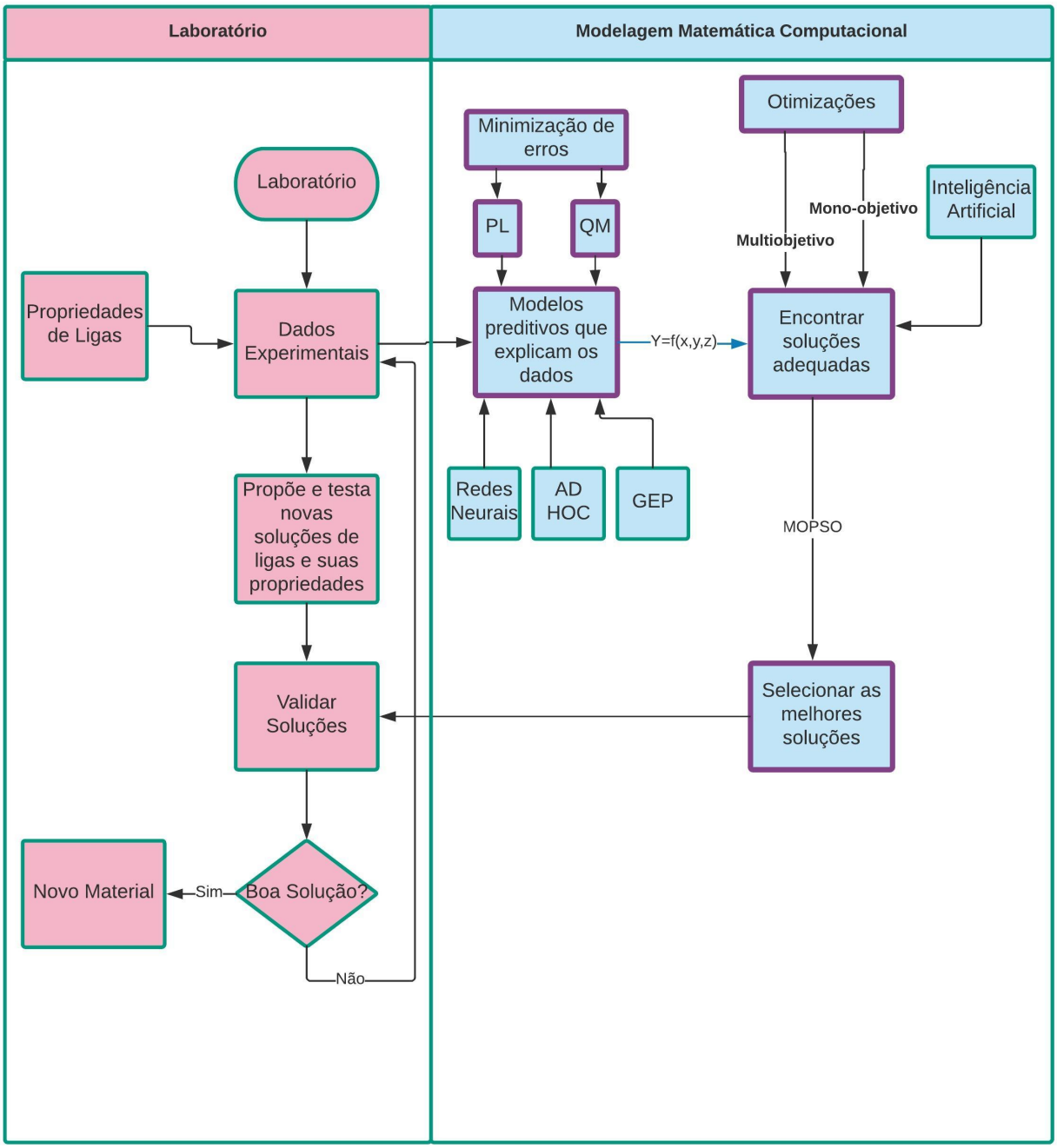

Figura 3.1: Metodologia de ajuste e otimização de modelos. Fonte própria

3.3 Aplicação da metodologia às ligas de Titânio

3.3.1 Contextualização do problema de tratamento das ligas de Titânio

Usaremos a mesma contextualização apresentada por Rafieerad et al. (2017), onde uma metodologia baseada em um modelo de programação multiobjetivo com resolução via algoritmo PSO multiobjetivo é proposta para melhorar as propriedades mecânicas, tribológicas, corrosivas e de bioatividade *in-vitro* de nanotubos de óxido misto em implantes Ti-6Al-7Nb. Segundo Rafieerad et al. (2017) as ligas de titânio, através do processo de *deposição física por vapor* (PVD) têm suas características melhoradas e ajustadas para serem utilizadas como biomateriais. Os parâmetros de entrada do processo PVD são as correntes DC Bias, DC Power e o fluxo de argônio enquanto que os parâmetros de saída são HV e AD. Nos modelos matemáticos as suas variáveis independentes são os parâmetros de entrada do PVD, ou seja, x_1 = DC Power, x_2 = fluxo de argônio e x_3 = DC Bias. A liga de Titânio recebe um revestimento de outros materiais melhorando a força de adesão (AD) e a dureza (HV) da liga revestida, aumentando a sua bio funcionalidade. Simultaneamente, os modelos de AD e HV da liga devem ser otimizadas. A metodologia *response surface methodology* (RSM) não é suficiente para a versão multiobjetivo dos modelos matemáticos. Mais detalhes sobre a RSM é fornecido por Myers et al. (2009).

Segundo Rafieerad et al. (2016) os modelos para força de adesão e dureza são ajustados por *genetic expression programming* (GEP), já em Rafieerad et al. (2017) os modelos são ajustados por minimização do erro absoluto usando heurística, ou seja, aplicando um algoritmo PSO mono-objetivo de dimensão 10. A otimização multiobjetivo de tais modelos e a seleção das melhores soluções é feita por algoritmos de metaheurística como o MOPSO. Em nosso trabalho, calculamos os coeficientes dos modelos pela minimização do erro absoluto (abordagem por programação linear) e por quadrados mínimos, pois estas abordagens são determinísticas e a abordagem por programação linear pode substituir o ajuste por GEP, bastando escolher a mesma função de ajuste em ambas as abordagens. Devido a ajustarmos os mesmos modelos por técnicas diferentes obtemos dois para a força de adesão e dois modelos para a dureza.

3.3.2 Modelos matemáticos aproximados para ligas de Titânio

Rafieerad et al. (2017) propõem os modelos da força de adesão e dureza através da técnica RSM. Em resumo, a técnica consiste em aproximar modelos matemáticos com várias variáveis por polinômios de Taylor multivariáveis de segunda ordem e calcular os coeficientes do polinômio de tal forma que o erro seja mínimo, podendo empregar assim o método dos quadrados mínimos. O polinômio que calcula a força de adesão $Y'(x_1, x_2, x_3)$ possui 10 coeficientes, porém os experimentos descritos no artigo oferecem apenas 9 pontos

						Normalized values				
No.	DC Power	Argon Rate	DC Bias	Adhesion	Hardness	DC Power	Argon Flow Rate	DC Bias	Adhesion	Hardness
1	150	20	30	877.03	258	1.0	1.0	1.0	1.090	1.252
2	150	40	60	1077.6	290	1.0	1.5	1.5	1.261	1.563
3	150	60	90	1432.07	247	1.0	2.0	2.0	1.562	1.145
4	250	20	60	1392.78	270	1.5	1.0	1.5	1.529	1.368
5	250	40	90	1277.76	276	1.5	1.5	2.0	1.431	1.427
6	250	60	30	770.42	232	1.5	2.0	1.0	1.000	1.000
7	**350**	**20**	**90**	**1946.35**	**335**	**2.0**	**1.0**	**2.0**	**2.000**	**2.000**
8	350	40	30	1503.63	297	2.0	1.5	1.0	1.623	1.631
9	350	60	60	1658.31	264	2.0	2.0	1.5	1.755	1.310

Figura 3.2: Dados do experimento. Fonte Rafieerad et al. (2017)

(resultados de experimentos), sendo então proposto o calculo dos coeficientes do polinômio utilizando o método do menor desvio absoluto. Abaixo seguem as equações do modelo preditivo apresentadas por Rafieerad et al. (2017) segundo a metodologia RSM

$$Y' = \sum_{i=1}^{n-1}\sum_{j=2}^{n}\beta_{ij}X'_iX'_j + \sum_{i=1}^{n}\beta_{ii}X'^2_i + \sum_{i=1}^{n}\beta_iX'_i + \beta_0$$
$$X' = a + \frac{(X - X_{min})(b-a)}{X_{max} - X_{min}},$$

sendo que X' e Y' indicam que os dados foram normalizados. Os coeficientes do polinômio são calculados minimizando a função $F(\beta) = \sum_{i=1}^{n}|Y'_i(observado) - Y'_i(\beta)(estimado)|$. Um modelo quadrático genérico para a força de adesão e a dureza *vickers* pode ser proposto como segue.

$$HV = \beta_0 + \beta_1x_1 + \beta_2x_2 + \beta_3x_3 + \beta_4x_1x_2 + \beta_5x_1x_3 + \beta_6x_2x_3 + \beta_7x_1^2 + \beta_8x_2^2 + \beta_9x_3^2$$
$$AD = \beta_0 + \beta_1x_1 + \beta_2x_2 + \beta_3x_3 + \beta_4x_1x_2 + \beta_5x_1x_3 + \beta_6x_2x_3 + \beta_7x_1^2 + \beta_8x_2^2 + \beta_9x_3^2,$$

sendo que os coeficientes β não são necessariamente iguais em HV e AD. Esta é uma notação genérica da metodologia RSM.

3.3.2.1 Cálculo dos coeficientes utilizando um Problema de Programação Linear (PL)

A matriz A abaixo apresenta os dados de 9 pontos experimentais. As linhas de A são da forma $(1, x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1^2, x_2^2, x_3^2)$, onde cada ponto do experimento gera uma linha de A. As colunas da matriz são os termos dos modelos sem os coeficientes.

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1.5 & 1.5 & 1.5 & 1.5 & 2.25 & 1 & 2.25 & 2.25 \\ 1 & 1 & 2 & 2 & 2 & 2 & 4 & 1 & 4 & 4 \\ 1 & 1.5 & 1 & 1.5 & 1.5 & 2.25 & 1.5 & 2.25 & 1 & 2.25 \\ 1 & 1.5 & 1.5 & 2 & 2.25 & 3. & 3. & 2.25 & 2.25 & 4 \\ 1 & 1.5 & 2 & 1 & 3. & 1.5 & 2 & 2.25 & 4 & 1 \\ 1 & 2 & 1 & 2 & 2 & 4 & 2 & 4 & 1 & 4 \\ 1 & 2 & 1.5 & 1 & 3. & 2 & 1.5 & 4 & 2.25 & 1 \\ 1 & 2 & 2 & 1.5 & 4 & 3. & 3. & 4 & 4 & 2.25 \end{bmatrix}$$

Sejam $B = (\beta_0, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7, \beta_8, \beta_9)^T$ o vetor de coeficientes que queremos calcular e $Y_{AD} = (1.090, 1.261, 1.562, 1.529, 1.431, 1.000, 2.000, 1.623, 1.755)^T$ o vetor de dados experimentais de AD. Utilizando a abordagem de ajuste de curvas descrita por Arenales et al. (2015), o seguinte PL pode ser formulado e resolvido para que os coeficientes sejam calculados.

$$\begin{aligned} &\text{Minimizar} \quad && z = \sum_{i=1}^{9}(e_i^+ + e_i^-) \\ &\text{Sujeito a} \quad && A.B^T + e^+ - e^- = Y_{AD}. \end{aligned}$$

Com o cálculo da solução B^* deste PL, obtemos

$$\begin{aligned} AD = 0.617619 - 0.422x_2 + 0.698x_3 - 0.767238x_1x_2 + 0.183238x_1x_3 \\ +0.787238x_2x_3 + 0.520381x_1^2 + 0.112381x_2^2 - 0.639619x_3^2. \end{aligned} \tag{3.3}$$

Os coeficientes de HV também são calculados com o uso de um PL, ou seja, minimizamos o erro absoluto e encontramos os coeficientes para o modelo HV resolvendo o seguinte PL construído a partir dos dados experimentais:

$Y_{HV} = (1.252, 1.563, 1.145, 1.368, 1.427, 1.000, 2.000, 1.631, 1.310)^T$.

$$\begin{aligned} &\text{Minimizar} \quad && z = \sum_{i=1}^{9}(e_i^+ + e_i^-) \\ &\text{Sujeito a} \quad && A.B^T + e^+ - e^- = Y_{HV}. \end{aligned}$$

Com o cálculo da solução B^* deste PL, obtemos

$$\begin{aligned} HV = -0.733381 - 1.58267x_1 + 4.47367x_2 + 0x_3 - 0.995238x_1x_2 + 0.788571x_1x_3 \\ +0.229905x_2x_3 + 0.759048x_1^2 - 1.17229x_2^2 - 0.515619x_3^2. \end{aligned} \tag{3.4}$$

Veja o Apêndice B para mais detalhes sobre os cálculos dos coeficientes dos modelos AD e HV. Segundo Rafieerad et al. (2017), os modelos matemáticos preditivos podem ter a sua precisão verificada pelo cálculo dos seguintes coeficientes: erro médio absoluto em

porcentagem (MAPE), erro quadrático médio (RMSE), U-statistic, R-quadrado, acurácia (A). Os cálculos e as fórmulas dos coeficientes para verificar a precisão estão descritos no Apêndice C. Utilizamos a mesma regra de Rafieerad et al. (2017) para verificar a precisão dos nossos modelos, assim conseguimos indicadores idênticos para uma possível comparação.

Modelo AD	
Erro absoluto	0
MSE	0
RMSE	0
MAPE(%)	0
U-statistic	0
A(%)	100
R2(%)	100

Tabela 3.1: Desempenho do modelo AD via PL

Modelo HV	
Erro absoluto	0
MSE	0
RMSE	0
MAPE(%)	0
U-statistic	0
A(%)	99.999
R2(%)	100

Tabela 3.2: Desempenho do modelo HV via PL

3.3.2.2 Cálculo dos coeficientes utilizando o método dos quadrados mínimos (QM)

Teorema 8 (Existência de quadrados mínimos)**.** *Dado qualquer sistema linear $Ax = b$, o sistema normal associado $A^TAx = A^Tb$ é consistente, e todas as soluções do sistema normal são soluções de mínimos quadrados de $Ax = b$. Além disso, se W for espaço coluna de A e x uma solução de mínimos quadrados qualquer de $Ax = b$, então a projeção ortogonal de b em W é $proj_w b = Ax$.*

Teorema 9 (Unicidade de quadrados mínimos)**.** *Se A for uma matriz $m \times n$ com vetores coluna linearmente independentes, então dada qualquer vetor b de tamanho $m \times 1$, o sistema linear $Ax = b$ tem uma única solução de mínimos quadrados. Esta solução é dada por $x = (A^TA)^{-1}A^Tb$.*

Anton and Rorres (2010) apresenta a existência e unicidade da solução para o método QM via Teorema 8 e 9, sendo que o último garante a unicidade da solução caso os vetores coluna da matriz A sejam linearmente independentes. Em nossa abordagem, a última coluna da matriz A, após redução de linhas, é combinação linear das outras colunas, portanto existem mais de uma solução QM para o nosso problema. Os coeficientes podem ser obtidos pela seguinte solução QM.

$$\beta = (A^T A)^{-1} A^T Y_{AD} \tag{3.5}$$

Podemos observar abaixo que existem várias soluções QM (uma para cada β_0) e, consequentemente, vários modelos podem ser derivados nesta estratégia.

$$\beta = \begin{bmatrix} 0 \\ 0.864667 \\ 0.442667 \\ -0.166667 \\ -1.26133 \\ 0.677333 \\ 1.28133 \\ 0.273333 \\ -0.134667 \\ -0.886667 \end{bmatrix} + \begin{bmatrix} 1 \\ -1.4 \\ -1.4 \\ 1.4 \\ 0.8 \\ -0.8 \\ -0.8 \\ 0.4 \\ 0.4 \\ 0.4 \end{bmatrix} \beta_0$$

Conforme o trabalho de Rafieerad et al. (2017), está disponível 9 pontos experimentais para o cálculo dos 10 coeficientes dos modelos, então temos um grau de liberdade neste ajuste. Se o ponto $AD(0,0,0) = 0$ for adicionado aos dados experimentais e calcularmos os coeficientes do modelo por QM obtemos um novo modelo para AD como segue.

$$\begin{aligned} AD = 0.864667x_1 + 0.442667x_2 - 0.166667x_3 - 1.26133x_1x_2 + 0.677333x_1x_3 \\ +1.28133x_2x_3 + 0.273333x_1^2 - 0.134667x_2^2 - 0.886667x_3^2. \end{aligned} \tag{3.6}$$

É de se esperar que quando os parâmetros do processo de revestimento forem iguais a zero não haverá revestimento, logo AD e HV que são decorrentes do processo de revestimento serão iguais a zero. O que torna natural incluir o ponto $AD(0,0,0) = 0$ aos dados experimentais.

A forma matricial para o cálculo dos coeficientes de HV é dada por

$$\beta = (A^T A)^{-1} A^T Y_{HV}$$

Da mesma forma que procedemos no cálculo de AD, incluímos o ponto $HV(0,0,0) = 0$ nos dados experimentais e calcularmos os coeficientes do modelo por QM e obtemos um

Modelo *AD*	
Erro absoluto	0
MSE	0
RMSE	0
MAPE(%)	0
U-statistic	0
A(%)	100
R2(%)	100

Tabela 3.3: Desempenho do modelo AD via QM

novo modelo para HV como segue.

$$\begin{aligned} HV = 0 - 2.6094x_1 + 3.44693x_2 + 1.02673x_3 - 0.408533x_1x_2 + 0.201867x_1x_3 \\ -0.3568x_2x_3 + 1.0524x_1^2 - 0.878933x_2^2 - 0.222267x_3^2 \end{aligned} \tag{3.7}$$

Modelo HV	
Erro absoluto	0
MSE	0
RMSE	0
MAPE(%)	0
U-statistic	0
A(%)	100
R2(%)	100

Tabela 3.4: Desempenho do modelo HV via QM

Rafieerad et al. (2017) sugerem um modelo matemático para a dureza HV como um polinômio de Taylor de segunda ordem com sete termos, além do mais o experimento oferece 9 pontos, então calculamos os coeficientes β do modelo $HV = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_1^2 + \beta_5 x_2^2 + \beta_6 x_3^2$ por QM e utilizamos os 9 pontos documentados por eles. Este cálculo foi realizado apenas para comparações com os resultados prévios descritos em Rafieerad et al. (2017). No entanto, verificamos que os modelos mais indicados e mais precisos são aqueles evidenciados no Capítulo 4 de experimentos computacionais. As linhas da matriz A_1 são da forma $(1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2)$ e contem os dados experimentais.

$$A_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1.5 & 1.5 & 1 & 2.25 & 2.25 \\ 1 & 1 & 2 & 2 & 1 & 4 & 4 \\ 1 & 1.5 & 1 & 1.5 & 2.25 & 1 & 2.25 \\ 1 & 1.5 & 1.5 & 2 & 2.25 & 2.25 & 4 \\ 1 & 1.5 & 2 & 1 & 2.25 & 4 & 1 \\ 1 & 2 & 1 & 2 & 4 & 1 & 4 \\ 1 & 2 & 1.5 & 1 & 4 & 2.25 & 1 \\ 1 & 2 & 2 & 1.5 & 4 & 4 & 2.25 \end{bmatrix}$$

$$\beta = (A_1^T A_1)^{-1} A_1^T Y_{HV}$$
$$Y_{HV} = [1.252, 1.563, 1.145, 1.368, 1.427, 1.000, 2.000, 1.631, 1.310]^T$$
$$HV = 1.32067 - 2.295x_1 + 1.94567x_2 + 0.283667x_3 + 0.874x_1^2 - 0.778x_2^2 - 0.018x_3^2$$

Nesta seção nos preocupamos com o ajuste de modelos matemáticos para AD e HV. Na próxima seção vamos detalhar questões e explicações que envolvam o algoritmo MOPSO, que é de grande importância na otimização dos modelos que apresentamos. Questões como a análise de dominância que são fundamentais para a implementação do algoritmo, os detalhes de como o algoritmo pesquisa soluções e as questões de como o algoritmo consegue uma aproximação da fronteira de Pareto.

3.3.3 Algoritmo enxame de partículas multiobjetivo

3.3.3.1 Análise de dominância para implementação do algoritmo MOPSO

No MOPSO temos duas funções objetivo, com 3 variáveis cada, e nove partículas que varrem o espaço de busca contido em $\mathbb{R}^3$. O espaço de busca está limitado por (x_{min}, x_{max}), (y_{min}, y_{max}) e (z_{min}, z_{max}) nas direções x, y e z, respectivamente. Em cada iteração do algoritmo selecionamos as posições das partículas que são soluções não dominadas. Uma partícula p_i é comparada com as outras partículas do conjunto $p = \{p_1, p_2, \ldots, p_{i-1}, p_{i+1}, \ldots, p_9\}$. Se na comparação da partícula i com as demais partículas verificarmos que qualquer uma das condições do teste computacional de dominância ($p_i.f_1 = p_k.f_1$ e $p_i.f_2 < p_k.f_2$), ou ($p_i.f_1 < p_k.f_1$ e $p_i.f_2 = p_k.f_2$), ou ($p_i.f_1 < p_k.f_1$ e $p_i.f_2 < p_k.f_2$), a partícula é dominada e consequentemente não é inserida no repositório de posições não dominadas. No início da execução do algoritmo as 9 partículas são inicialmente consideradas não dominadas, em seguida se a partícula é reprovada pelo teste de dominância, ela não será inserida no repositório. Veja os casos possíveis de dominância entre as partículas na Tabela 3.5.

Resultados de Comparações $\bigoplus = \{=, <, >\}$			
condição i	$p_i.f_1 \bigoplus p_k.f_1$	$p_i.f_2 \bigoplus p_k.f_2$	solução
1	$=$	$=$	não dominada
2	$=$	$>$	não dominada
3	$=$	$<$	**dominada**
4	$>$	$=$	não dominada
5	$>$	$>$	não dominada
6	$>$	$<$	não dominada
7	$<$	$=$	**dominada**
8	$<$	$>$	não dominada
9	$<$	$<$	**dominada**

Tabela 3.5: Análise de dominância para maximização

O Resultado das comparações entre as partículas conforme a tabela é descrito pelo operador $\bigoplus$, sendo possível o seguinte conjunto de resultados $\{=, >, <\}$. Dizemos que uma partícula é não dominada no conjunto se não existe neste conjunto alguma outra partícula que a domine. Como o algoritmo PSO multiobjetivo foi implementado neste trabalho com duas funções objetivo, expressamos a relação de dominância entre as partículas como $D_S^{p_i}(\oplus, \oplus) = \{0, 1\}$, onde 0 significa que a partícula i é dominada e 1 significa que a partícula i é não dominada quando ela é comparada com todas as outras partículas do conjunto S. Consequentemente temos os seguintes resultados ao verificarmos a relação de dominância de uma partícula $\{D(=,=) \to 1, D(=,>) \to 1, D(=,<) \to 0,\ D(>,=) \to 1, D(>,>) \to 1, D(>,<) \to 1, D(<,=) \to 0, D(<,>) \to 1, D(<,<) \to 0\}$.

Na implementação do algoritmo MOPSO, se uma das condições do teste de dominância (Condições 3, 7 e 9 da Tabela 3.5) for satisfeita para partícula i ela é dominada. É importante ressaltar que o papel das partículas é pesquisar o espaço de busca se movimentando aleatoriamente e adicionar posições não dominadas ao repositório. O repositório acumula posições que permanecem não dominadas durante todas as iterações do algoritmo. Portanto, no final das iterações, este repositório contém uma aproximação da fronteira de Pareto.

3.3.3.2 Varredura do espaço de busca pelo algoritmo MOPSO

No PSO a varredura do espaço de busca, ou seja, a atualização das posições $(x_{i,1}, x_{i,2}, \ldots, x_{i,d})$ das partículas são feitas através das equações

$$v_{i,d}(t+1) = wv_{i,d}(t) + c_1 r_1(p_{i,d} - x_{i,d}(t)) + c_2 r_2(g_{i,d} - x_{i,d}(t)), \tag{3.8}$$

$$x_{i,d}(t+1) = x_{i,d}(t) + v_{i,d}(t+1). \tag{3.9}$$

É importante verificar inicialmente como as partículas varrem o espaço de decisão, pois este espaço é mapeado para o espaço objetivo. Com os recursos computacionais visuais do software *Wolfram Mathematica* é possível verificar visualmente (Figura 3.3) se as buscas

terão sucesso em encontrar a fronteira de Pareto. O algoritmo MOPSO é uma ferramenta de busca e pode ser executado várias vezes, sendo possível ainda após algumas execuções restringirmos o espaço de busca para que as soluções não dominadas sejam encontradas mais rapidamente.

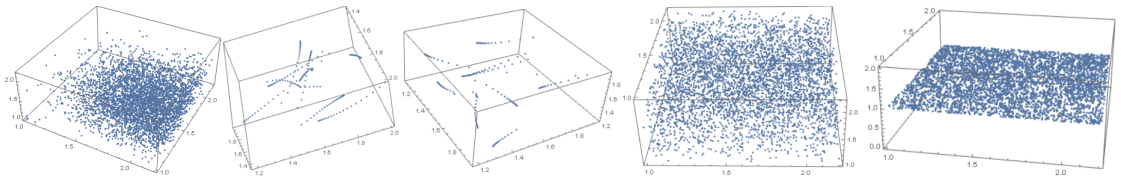

Figura 3.3: Varreduras do espaço de busca pelo MOPSO. Fonte própria

3.3.3.3 Fronteira de Pareto pesquisada pelo algoritmo MOPSO

Aqui vamos dar uma ideia de como as partículas se deslocam, e uma noção da quantidade de posições do espaço que dominam a posição atual da partícula que se desloca, pois na fronteira de Pareto nenhum dos pontos domina um outro. Assim, parece razoável que o movimento de uma dada partícula deva ocorrer na direção de posições que dominam a posição atual desta partícula. Esta seção também ajuda a compreender os testes com o algoritmo no Capítulo 4.

Vamos observar a Figura 3.4 para compreendermos como as partículas podem atingir a fronteira de Pareto. Conforme proposto no algoritmo PSO, a partícula ao se mover no espaço move em uma direção que é combinação de dois vetores $\overrightarrow{p}$ e $\overrightarrow{g}$, sendo possível priorizar a participação de cada vetor por meio de coeficientes, podemos então dizer que a posição da partícula ao longo do tempo é $\overrightarrow{x}(t+1) = \overrightarrow{x}(t) + \overrightarrow{v}(t) + k_1\overrightarrow{p} + k_2\overrightarrow{g}_{Randomico}$, lembrando que estes vetores variam ao longo do movimento.

Podemos observar na Figura 3.4 que quando a partícula x_i se desloca ela pode atingir um dos 4 quadrantes, se atingir o quadrante R superior, a sua nova posição (f_1, f_2) no espaço objetivo domina a posição anterior. O vetor $\overrightarrow{p}$ é calculado como $\overrightarrow{p} = \overrightarrow{x}_{anterior} - \overrightarrow{x}_{atual}$ se a posição anterior domina a posição atual ou $\overrightarrow{p} = \overrightarrow{0}$ se a posição atual domina a anterior. A probabilidade de que a partícula atinja uma próxima posição (caso o movimento seja completamente aleatório) que domine a posição atual é cerca de 25% (região R superior) e cerca de 50% de atingir uma próxima posição onde a nova posição não domine a posição atual e vice versa (regiões ND) e há também a possibilidade da partícula se movimentar para uma nova localidade onde a nova posição é dominada pela posição anterior (região R inferior) com probabilidade de 25%. Estas probabilidades dependem da forma do espaço objetivo e da posição da partícula, é possível ter uma ideia destas probabilidades olhando a Figura 3.5. Observe que se uma partícula x_i se encontra na posição A da Figura 3.5 e tiver seu vetor $\overrightarrow{g}$ apontado para a região demarcada é muito provável que ela ocupe uma nova posição que domine a posição anterior.

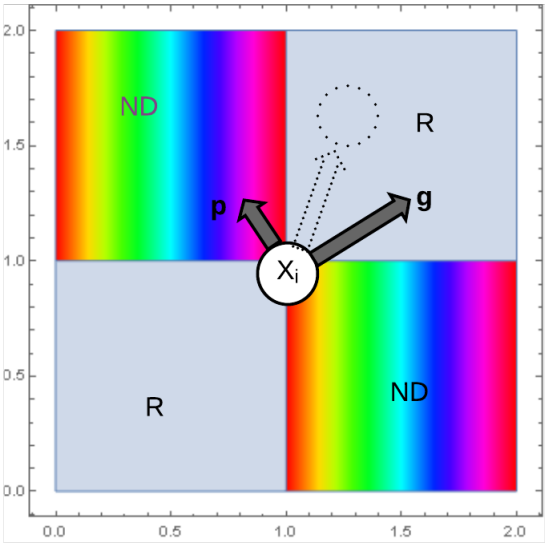

Figura 3.4: Estudo do movimento de partículas quanto a dominância. Fonte própria

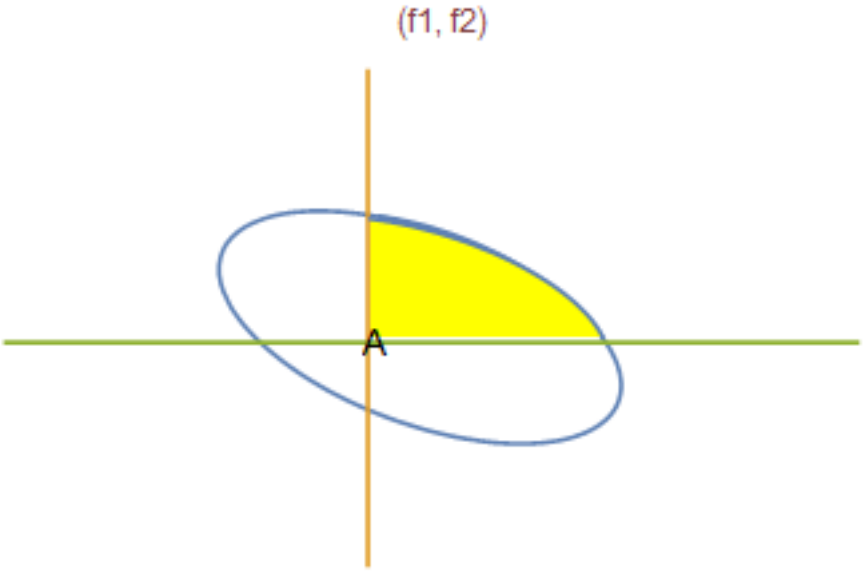

Figura 3.5: Probabilidade do movimento em direção a curva de Pareto. Fonte própria.

3.3.3.4 Melhorando a convergência do algoritmo para fronteira de Pareto

Conforme veremos nos testes, no capítulo 4, foi possível melhorar a convergência do algoritmo alterando a direção do vetor $\overrightarrow{g}$ $(gbest - x_{atual})$ fazendo com que ele fique restrito a região entre os gradientes das funções objetivo. Segundo as regras do algoritmo a posição

gbest (líder) deve ser uma posição armazenada no repositório que domine a posição atual da partícula que está pesquisando posições, caso não haja no repositório uma posição que atenda a este requisito podemos escolher aleatoriamente qualquer outra posição do repositório. Convêm observar que em cada iteração do algoritmo posições pesquisadas pelas partículas saem e entram no repositório, uma posição que seja indicada na iteração i como a posição gbest poderá na próxima iteração ser uma posição dominada pela posição atual de uma partícula, esta situação faz que o vetor $\overrightarrow{g}$ ora oriente o movimento de maneira positiva ora de maneira negativa. A estratégia de escolha aleatória do vetor $\overrightarrow{g}$ tem uma boa probabilidade de orientar positivamente o movimento na Iteração i, veja a Figura 3.4 para ter noção da quantidade de posições que venham dominar a posição atual da partícula x_i em movimentos subsequentes, este fato influencia positivamente o movimento em direção a fronteira de Pareto. O repositório armazena posições do espaço de busca pesquisadas pelas partículas para que no final do algoritmo tenhamos uma aproximação da fronteira de Pareto.

Pelas regras do algoritmo é enfatizado movimentos na direção de novas posições que dominam a posição atual, esta heurística é observada nas duas direções que geram a nova posição, direção $\overrightarrow{g}$ e direção $\overrightarrow{p}$, sendo assim ao escolher a direção do vetor $\overrightarrow{g}$ como uma direção entre $\nabla f1$ e $\nabla f2$, (Figura 3.6), estamos deslocando para novas posições que dominam a posição atual, esta afirmação pode ser verificada examinando o espaço objetivo (Figura 3.4), pois ao deslocarmos no quadrante R superior estamos melhorando as duas funções objetivo simultaneamente e no espaço de decisão como as partículas devem se deslocar para que as funções objetivo melhorem simultaneamente? Respondendo a esta pergunta, propomos neste trabalho que a partícula deve se orientar pelos vetores gradientes das funções objetivo, sendo então a nova posição da partícula dada pelas equações que propomos como segue

$$\overrightarrow{x}_{(t+1)} = w_{(t)}\overrightarrow{v}_{(t)} + \overrightarrow{x}_{(t)} + c_2 rand(0,1)(\nabla f1 + \nabla f2) + k_1\overrightarrow{p}, \tag{3.10}$$

$$\overrightarrow{g}_{Convergente} = (\nabla f1 + \nabla f2). \tag{3.11}$$

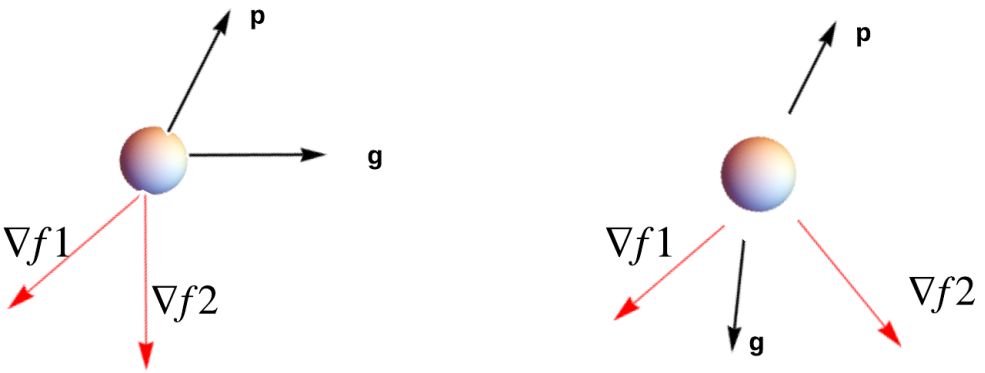

Figura 3.6: Movimento orientado. Fonte própria

3.3.3.5 Dinâmica de atualizações do repositório

Durante a execução do algoritmo o repositório é atualizado com posições que não dominam umas as outras, no final do algoritmo as posições armazenadas são uma aproximação da fronteira de Pareto. Convêm observar um possível cenário onde na iteração t do algoritmo já existam posições armazenadas no repositório, veja a Figura 3.7, na próxima iteração as partículas se movimentam segundo as regras propostas pelo algoritmo e a partícula x_i se desloca então para uma posição melhor quanto a dominância que todas as posições armazenadas, na próxima iteração t+1 por hipótese não existem no repositório posições armazenadas que dominem a posição atual, consequentemente o vetor $\overrightarrow{g}$ que orientará o próximo movimento da partícula x_i deverá apontar então para uma posição do repositório pior que a posição da partícula x_i, este cenário não é positivo para a convergência do algoritmo, a escolha aleatório da direção $\overrightarrow{g}$ pode ser uma estratégia melhor para encontrar soluções que dominem a posição atual que a estratégia de buscar soluções no repositório, principalmente quando as partículas estão em regiões onde a probabilidade de se mover para uma região melhor é alta. Neste cenário ruim para o repositório como guia do movimento, há poucas posições armazenadas que não dominam umas as outras, não refletindo ainda qualquer aproximação da fronteira de Pareto. O papel ou funcionalidade adequada do repositório nos algoritmos pode ser destacado como local de armazenamento de soluções não dominadas durante as iterações do algoritmo, o algoritmo PASA usa o conceito de repositório apenas para armazenar soluções não dominadas. Para ressaltar o papel adequado do repositório verifiquemos a seção 3.3.3.4, a estratégia que melhora a convergência do algoritmo não usa o repositório como guia do movimento, ou seja, as posições que dominam a posição atual de uma partícula são capturadas da região entre os gradientes do espaço de busca e não do repositório, veja as Equações 3.10 e 3.11

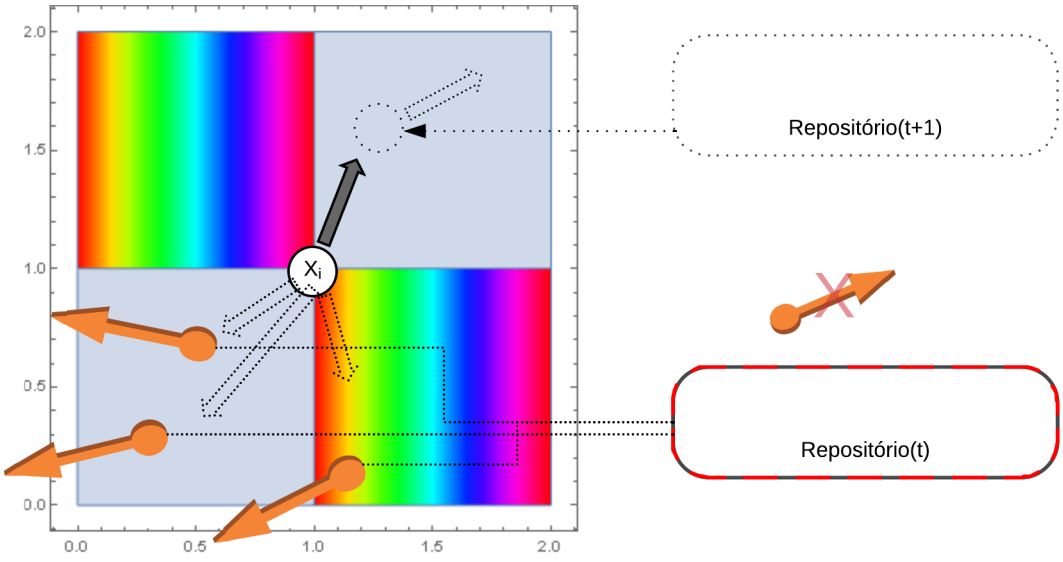

Figura 3.7: Repositório. Fonte própria

3.3.3.6 Algoritmo MOPSO e termos da literatura

Apresentamos nos itens anteriores da Seção 3.3.3.2 uma análise das principais questões sobre o algoritmo MOPSO, analisamos a convergência do algoritmo, analisamos o papel do repositório nos algoritmos MOPSO, analisamos o movimento das partículas no espaços de decisão e objetivo, nesta seção apresentamos o algoritmo MOPSO numa forma comum encontrada na literatura, embora existam várias adaptações deste algoritmo, aqui nesta seção apresentamos o algoritmo numa linguagem conceitual baseada nas origens do algoritmo (bio inspirado). Podemos observar o item *selecione o líder para a partícula* do Algoritmo 3.1, este item está relacionado ao vetor $\overrightarrow{g}$ apresentado nas seções anteriores e a posição g_i da equação de velocidades. A posição do líder aparece na literatura como gbest, já a melhor posição individual da partícula está relacionada ao vetor $\overrightarrow{p}$ apresentado nas seções de discussão e análise do algoritmo, a melhor posição da partícula é conhecida na literatura como pbest. É importante ressaltarmos o conceito de partícula, ou seja, um agente que faz buscas no espaço de decisão, este mesmo conceito aparece na literatura como indivíduo, agora computacionalmente uma partícula é uma estrutura de dados.

Algoritmo 3.1 Modelo do algoritmo PSO multiobjetivo

Inicialize o enxame contendo n- partículas;
Avalie as partículas
Determine as soluções não dominadas e armazene no repositório
Enquanto o número máximo de iterações não seja atingido
    Para cada partícula no enxame
        Selecione o líder para a partícula
        Atualize a velocidade de acordo com a regra
        Atualize a posição de acordo com a regra
        Avalie as funções objetivo para a partícula
        Atualize a melhor posição pbest da partícula conforme as regras:
            Se a nova posição domina pbest anterior **então** substitua pbest pela posição atual
            Se a nova posição é dominada pela posição pbest **então** não faça nada
            Senão escolha de maneira aleatória pbest entre a posição atual e a nova
    fim para
Adicione as partículas não dominadas ao repositório
Remova os membros dominados do repositório
fim enquanto
Para cada membro do repositório
    Imprima sua posição e valores das funções objetivo
fim

Capítulo 4

Experimentos computacionais

4.1 Noções gerais sobre a parte computacional

A Figura 4.1 facilita a compreensão das etapas da implementação computacional, vistas em detalhes na seção 4.3 e a compreensão dos experimentos numéricos. Os testes foram realizados com 500 iterações e 9 partículas por iteração, pois com este número de iterações e partículas obtemos até 100 pontos aproximados da fronteira de Pareto. Como resultado da execução do algoritmo MOPSO temos a aproximação da fronteira de Pareto apenas, os outros resultados como espaço de decisão, espaço objetivo e partículas não dominadas são resultados complementares que dão uma ideia do comportamento do algoritmo até a obtenção da aproximação da fronteira de Pareto. Podemos observar ainda que os modelos $F(x, y, z)$ e $G(x, y, z)$ são representações na figura de modelos a serem otimizados.

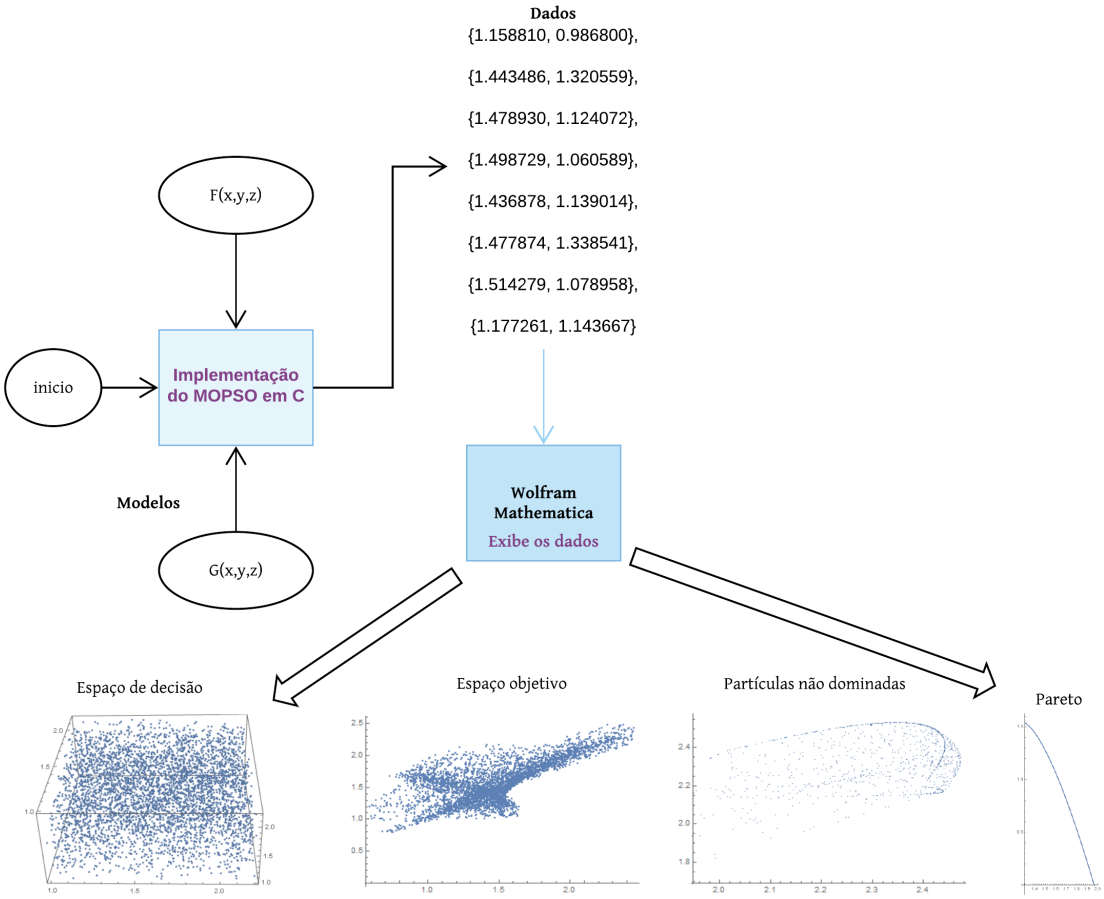

Figura 4.1: Etapas da implementação computacional. Fonte própria

4.2 Implementação do algoritmo PSO multiobjetivo

Nossa implementação do algoritmo PSO multiobjetivo, baseou-se no estuda da literatura, há algumas variações de implementação conforme verificado nos artigos, o algoritmo PSO multiobjetivo é uma adaptação do PSO mono-objetivo. Em cada iteração do algoritmo, partículas entram e saem do repositório, entram as partículas não dominadas por nenhuma outra no repositório e saem as partículas dominadas por quem entra no repositório. Uma das variações da implementação é sobre a escolha do líder. É relatado em trabalhos sobre algumas possibilidades de escolha, esta escolha afeta diretamente o comportamento de buscas do algoritmo, o líder está relacionado com o parâmetro *gbest* da equação de velocidades. O conceito de líder no algoritmo MOPSO não mantem todas as características deste mesmo conceito empregado no algoritmo PSO. No algoritmo PSO o vetor $\overrightarrow{g}$ de cada partícula em cada iteração do algoritmo aponta sempre para a mesma posição(gbest), enquanto este mesmo vetor $\overrightarrow{g}$ no algoritmo MOPSO aponta para posições

diferentes, espera-se que aponte para posições que dominem a posição atual da partícula.

Na implementação multiobjetivo o conceito de melhor e pior está relacionado ao conceito de dominância, o melhor domina, enquanto na implementação mono-objetivo o conceito de melhor ou pior está relacionado aos valores da função nos pontos pesquisados pelas partículas.

4.3 Construção de uma solução computacional

O código do algoritmo PSO foi escrito em linguagem C padrão C11. O motivo da escolha foi o histórico da linguagem, como velocidade, independência de outros recursos computacionais, não havendo necessidade de instalar recursos externos para que a solução computacional fosse executada e também não havendo necessidade de licenças de software. Para esta implementação computacional a linguagem C possibilitou uma implementação maleável (extensível) quanto a adição de novos modelos, em decorrência desta característica não é necessário alterar o código principal da solução computacional quando novos modelos vierem a ser testados.

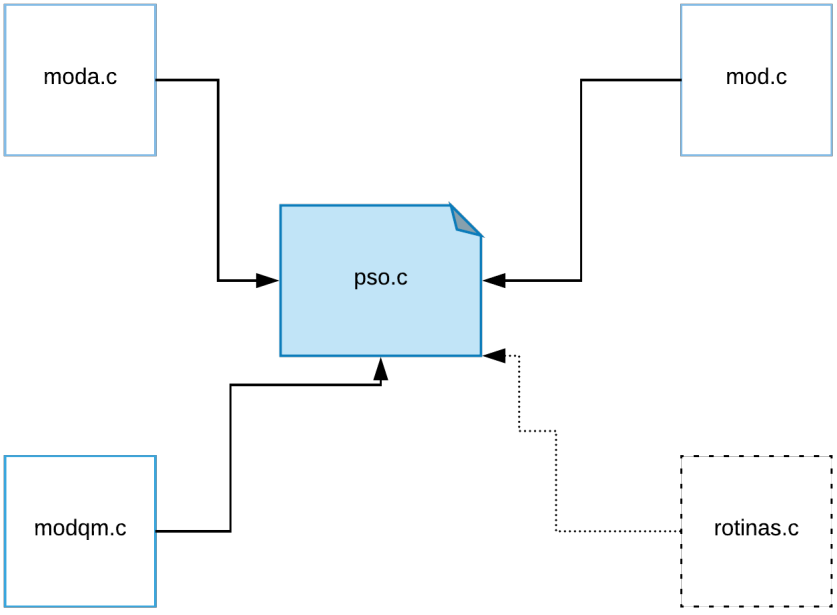

Figura 4.2: Organização do Codigo. Fonte própria

Quanto a implementação temos o arquivo *pso.c*, o arquivo principal e temos os arquivos *moda.c*, *mod.c* e *modqm.c*. Nestes arquivos estão as implementações dos modelos em linguagem C. No arquivo *moda.c* temos a implementação do modelos conforme descrito no artigo, no arquivo *mod.c* temos a implementação dos modelos ajustados por programação

linear, no arquivo *modqm.c* temos as implementações dos modelos ajustados por mínimos quadrados. Ponteiros de função são variáveis do tipo função. Quando executamos o programa principal, escolhemos um modelo a executar e testar, por meio de ponteiros de função as funções implementadas nos arquivos de modelos se ligam ao programa principal durante a compilação. Quando executamos a solução computacional alguns arquivos são gerados, *pareto.txt*, *naodominada.txt*, *particula.txt*, *f1xf2.txt*, estes arquivos apresentam os dados respectivamente da fronteira de Pareto, dados de posições pesquisadas pelas partículas onde não houve dominância entre estas partículas em cada iteração, dados de todas as posições pesquisadas pelas partículas, dados do espaço objetivo. Os dados gerados pela solução computacional são visualizados no software *Wolfram Mathematica* através de gráficos, conforme a Figura 4.1. Através do comando *ListPlot*, *ListPointPlot3D* aplicado aos dados visualizamos os gráficos. Dentro do arquivo *pso.c* temos ponteiros de funções onde podemos escolher para qual modelo implementado nosso ponteiro irá apontar. Para entender melhor a estrutura do código vamos observar a Figura 4.2. Quanto a compilação do código usamos os seguintes comandos

gcc -std=c11 -c mod.c

gcc -std=c11 -c modqm.c

gcc -Wall -pedantic-errors -ansi -std=c11 pso.c -o pso mod.o -lm

gcc -Wall -pedantic-errors -ansi -std=c11 pso.c -o pso modqm.o -lm

Conforme as instruções de compilação logo acima, compilamos previamente os arquivos com modelos. Quando formos executar o programa principal *pso.c* compilamos o arquivo *pso.c* e o ligamos ao modelo que escolhemos trabalhar.

Configurações de máquina para os testes: Nos testes e durante o desenvolvimento estamos usando uma máquina virtual Linux da *Amazon AWS*, Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz, 1 thread(s) por núcleo e 1 núcleo. Através do utilitário *Putty* e *winCP* acessamos a máquina na *Amazon*.

4.4 Resultados dos experimentos computacionais

Nas próximas seções vamos exibir os resultados dos testes para o modelo matemático da força de adesão e o modelo matemático da dureza. Os testes apresentam como resultado uma figura com o gráfico da fronteira de Pareto e a tabela de dados correspondente, onde os dados das tabelas estão ordenados. Nos testes temos uma variação dos parâmetros de entrada, variação específica do teste, vamos trabalhar com 2 intervalos diferentes de variações dos parâmetros de entrada. Os dados foram obtidos pela execução do algoritmo MOPSO. O Resultado dos testes será apresentado como um gráfico da fronteira de Pareto

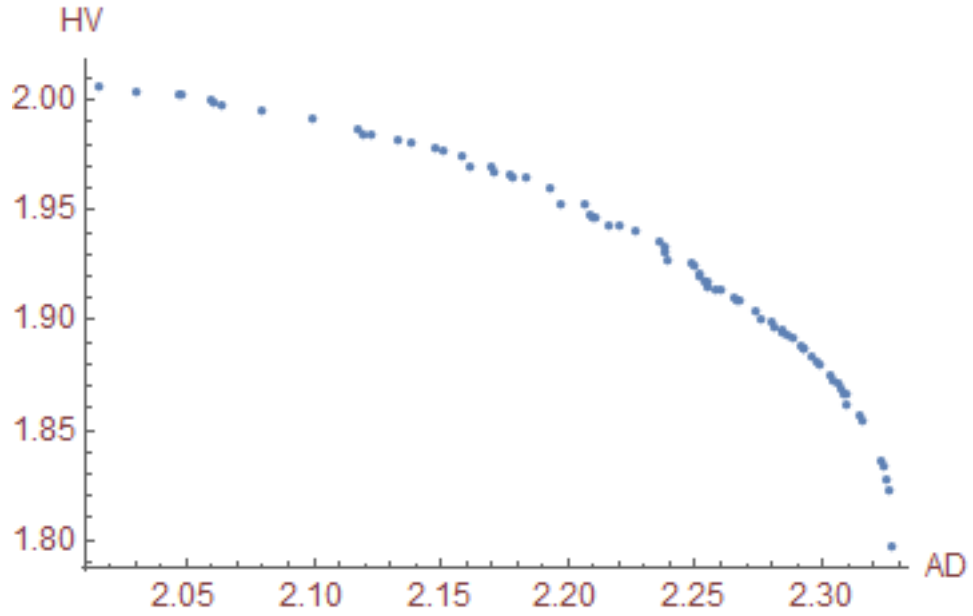

Figura 4.3: Fronteira de Pareto via QM - Tabela 4.1. Fonte própria

e uma tabela com os dados que geraram a curva de Pareto, estes dados são decorrentes da otimização dos modelos pelo algoritmo MOPSO. Vamos comparar resultados obtidos pelo *Wolfram Mathematica* e pelo algoritmo MOPSO. Após análise dos dados devemos dar prioridade a valores mais altos para HV e AD, pois nosso problema é de maximização simultânea dos objetivos.

4.4.1 Fronteira de Pareto e quadrados mínimos (QM)

Neste cenário é realizado testes para os modelos HV e AD por QM, ou seja, calculamos os erros dos coeficientes por QM, assim pudemos minimizar o erro nos coeficientes.

$$\begin{aligned} HV = -2.6094x_1 + 3.44693x_2 + 1.02673x_3 - 0.408533x_1x_2 + 0.201867x_1x_3 \\ -0.3568x_2x_3 + 1.0524x_1^2 - 0.878933x_2^2 - 0.222267x_3^2 \end{aligned} \tag{4.1}$$

$$\begin{aligned} AD = 0.864667x_1 + 0.442667x_2 - 0.166667x_3 - 1.26133x_1x_2 + 0.677333x_1x_3 \\ +1.28133x_2x_3 + 0.273333x_1^2 - 0.134667x_2^2 - 0.886667x_3^2 \end{aligned} \tag{4.2}$$

A Tabela 4.1 e a Figura 4.3 apresentam os resultados para os parâmetros de entrada x, y e z variando entre 1 e 2. Enquanto que a Tabela 4.2 e a Figura 4.4 dizem respeito ao intervalo variando entre 1 e 3.

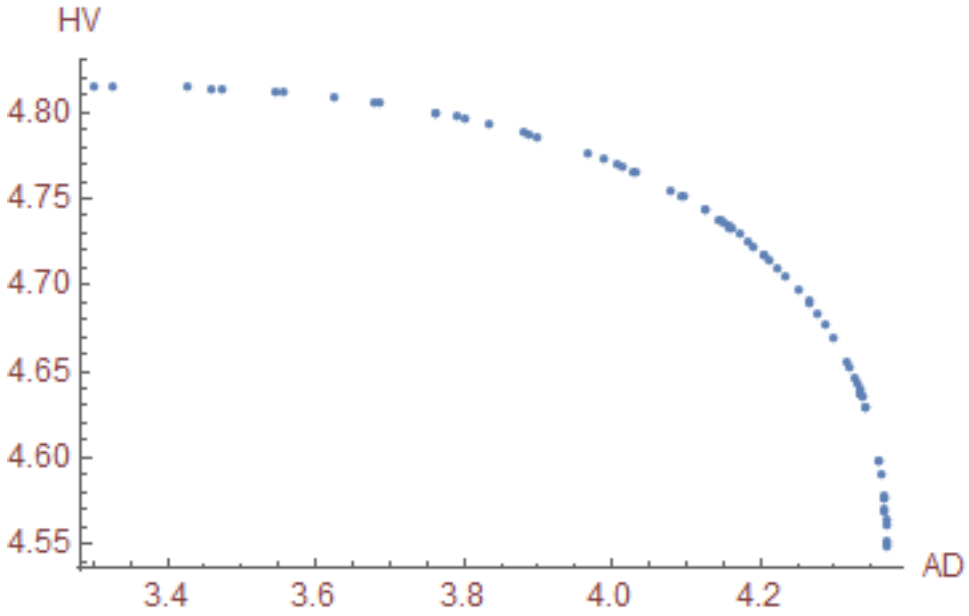

Figura 4.4: Fronteira de Pareto via QM - Tabela 4.2. Fonte própria

Intervalo $[1,2]$ **para os parâmetros** x**,** y **e** z			
Soluções	AD	HV	(x,y,z)
0	2.01512	2.00692	2.00000,1.07438,2.00000
12	2.02996	2.00335	2.00000,1.15575,2.00000
24	2.04716	2.00256	2.00000,1.10634,1.97162
33	2.04769	2.00237	2.00000,1.09970,1.96993
8	2.05975	1.99984	2.00000,1.13400,1.96146
10	2.06044	1.99871	2.00000,1.07347,1.95162
4	2.06429	1.99805	2.00000,1.15137,1.95856
20	2.12288	1.98426	2.00000,1.16008,1.87958
5	2.14723	1.97825	2.00000,1.10248,1.84284
13	2.15784	1.97484	2.00000,1.11621,1.82573
16	2.17037	1.96739	2.00000,1.06069,1.80998
21	2.17685	1.96674	2.00000,1.14112,1.78950
72	2.17822	1.96493	2.00000,1.15766,1.78352
31	2.18317	1.96479	2.00000,1.11689,1.78199
9	2.19238	1.96043	2.00000,1.12036,1.76364
36	2.20670	1.95236	2.00000,1.12820,1.73074
15	2.21548	1.94357	2.00000,1.15304,1.69590
11	2.22017	1.94258	2.00000,1.03606,1.73202
27	2.23610	1.93535	2.00000,1.07289,1.68663
17	2.23820	1.93102	2.00000,1.02344,1.70217
18	2.23885	1.92722	2.00000,1.00000,1.70823
6	2.24990	1.92522	2.00000,1.05950,1.66048
35	2.25118	1.91992	2.00000,1.00000,1.68538
25	2.25418	1.91802	2.00000,1.00000,1.67956
39	2.25811	1.91447	2.00000,1.08898,1.61233
26	2.26648	1.90961	2.00000,1.00000,1.65428
37	2.27355	1.90425	2.00000,1.00000,1.63859
14	2.28027	1.89884	2.00000,1.02536,1.60332
34	2.28077	1.89720	2.00000,1.03706,1.59048
22	2.28442	1.89501	2.00000,1.00000,1.61227
29	2.28814	1.89150	2.00000,1.00000,1.60250
40	2.29159	1.88806	2.00000,1.00000,1.59304
2	2.29215	1.88748	2.00000,1.00000,1.59144
32	2.29279	1.88682	2.00000,1.00000,1.58963
7	2.29815	1.88089	2.00000,1.00000,1.57365
30	2.30459	1.87282	2.00000,1.00000,1.55234
19	2.30573	1.87126	2.00000,1.00000,1.54828
3	2.30755	1.86865	2.00000,1.00000,1.54153
1	2.30873	1.86688	2.00000,1.00000,1.53700
38	2.30938	1.86171	2.00000,1.01126,1.51476
28	2.32371	1.83368	2.00000,1.00000,1.45571
23	2.32522	1.82706	2.00000,1.00000,1.44032

Tabela 4.1: Soluções de Pareto via QM - Figura 4.3

Intervalo $[1,3]$ **para os parâmetros** x**,** y **e** z			
Soluções	AD	HV	(x,y,z)
34	3.29933	4.81579	3.00000,1.00000,2.87320
11	3.32737	4.81577	3.00000,1.00000,2.85871
16	3.42849	4.81487	3.00000,1.00000,2.80478
29	3.45900	4.81432	3.00000,1.00000,2.78794
0	3.55522	4.81165	3.00000,1.00000,2.73292
8	3.62371	4.80878	3.00000,1.00000,2.69173
25	3.68534	4.80539	3.00000,1.00000,2.65303
17	3.76090	4.80003	3.00000,1.00000,2.60311
30	3.76315	4.79985	3.00000,1.00000,2.60158
15	3.79049	4.79753	3.00000,1.00000,2.58273
2	3.80259	4.79643	3.00000,1.00000,2.57424
19	3.88783	4.78736	3.00000,1.00000,2.51170
3	3.89964	4.78589	3.00000,1.00000,2.50261
10	4.00507	4.77006	3.00000,1.00000,2.41578
18	4.07868	4.75524	3.00000,1.00000,2.34742
22	4.12378	4.74403	3.00000,1.00000,2.30116
23	4.12693	4.74317	3.00000,1.00000,2.29778
39	4.14411	4.73830	3.00000,1.00000,2.27893
9	4.14624	4.73768	3.00000,1.00000,2.27654
28	4.15222	4.73588	3.00000,1.00000,2.26978
1	4.15931	4.73370	3.00000,1.00000,2.26165
32	4.16056	4.73331	3.00000,1.00000,2.26020
20	4.18409	4.72552	3.00000,1.00000,2.23207
6	4.19157	4.72286	3.00000,1.00000,2.22276
33	4.20474	4.71794	3.00000,1.00000,2.20586
38	4.23380	4.70584	3.00000,1.00000,2.16605
37	4.25239	4.69699	3.00000,1.00000,2.13829
14	4.27636	4.68389	3.00000,1.00000,2.09902
12	4.2977	4.67002	3.00000,1.00000,2.05955
27	4.32034	4.65186	3.00000,1.00000,2.01056
24	4.32634	4.64616	3.00000,1.00000,1.99576
35	4.32954	4.64292	3.00000,1.00000,1.98746
21	4.33547	4.63647	3.00000,1.00000,1.97115
5	4.34162	4.62900	3.00000,1.00000,1.95265
40	4.36004	4.59788	3.00000,1.00000,1.87923
31	4.36302	4.59011	3.00000,1.00000,1.86173
13	4.36678	4.57687	3.00000,1.00000,1.83260
26	4.36807	4.57024	3.00000,1.00000,1.81830
4	4.36895	4.56394	3.00000,1.00000,1.80490
7	4.36920	4.56157	3.00000,1.00000,1.79990
36	4.36978	4.54902	3.00000,1.00000,1.77381

Tabela 4.2: Soluções de Pareto via QM - Figura 4.4

4.4.2 Fronteira de Pareto e programação linear (PL)

Neste cenário realizamos os testes para os modelos HV e AD por PL, ou seja, calculamos os coeficientes dos modelos através de um problema de programação linear.

$$\begin{aligned} AD = 0.617619 - 0.422x_2 + 0.698x_3 - 0.767238x_1x_2 + 0.183238x_1x_3 \\ +0.787238x_2x_3 + 0.520381x_1^2 + 0.112381x_2^2 - 0.639619x_3^2 \end{aligned} \tag{4.3}$$

$$HV = -0.733381 - 1.58267x_1 + 4.47367x_2 + 0x_3 - 0.995238x_1x_2 + 0.788571x_1x_3 \\ +0.229905x_2x_3 + 0.759048x_1^2 - 1.17229x_2^2 - 0.515619x_3^2 \quad (4.4)$$

A Tabela 4.3 e a Figura 4.5 apresentam os resultados para os parâmetros de entrada x, y e z variando entre 1 e 2. Enquanto que a Tabela 4.4 e a Figura 4.6 dizem respeito ao intervalo variando entre 1 e 3.

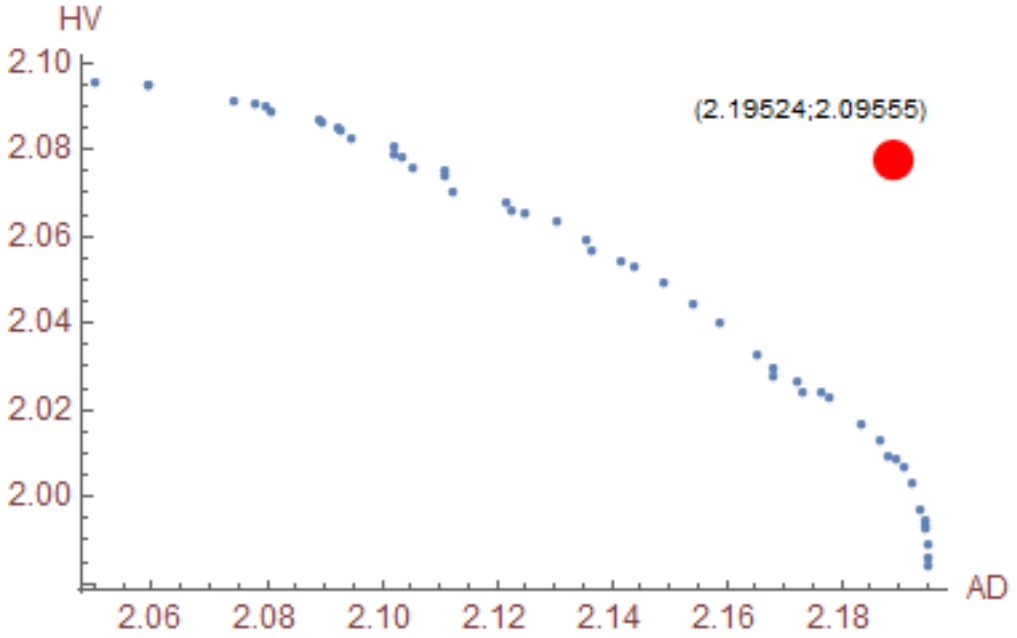

Figura 4.5: Fronteira de Pareto via PL - Tabela 4.3. Fonte própria

Intervalo $[1,2]$ **para os parâmetros** x**,** y **e** z			
Soluções	AD	HV	(x,y,z)
17	2.05000	2.09548	2.00000,1.22918,1.79833
46	2.05924	2.09477	2.00000,1.22320,1.76819
7	2.05947	2.09471	2.00000,1.22413,1.76602
26	2.07417	2.09137	2.00000,1.17976,1.76006
21	2.07769	2.09059	2.00000,1.17654,1.75009
36	2.07958	2.08998	2.00000,1.17280,1.74722
22	2.08919	2.08691	2.00000,1.17482,1.69963
2	2.08938	2.08641	2.00000,1.15521,1.73053
40	2.09248	2.08496	2.00000,1.17459,1.68114
44	2.09259	2.08443	2.00000,1.17722,1.67413
31	2.09449	2.08278	2.00000,1.17699,1.66142
25	2.10185	2.08102	2.00000,1.13643,1.70474
42	2.10204	2.07864	2.00000,1.11949,1.72843
13	2.10353	2.07807	2.00000,1.16006,1.64293
6	2.10548	2.07575	2.00000,1.15926,1.62886
28	2.11074	2.07535	2.00000,1.11326,1.70258
48	2.11083	2.0742	2.00000,1.14518,1.63296
14	2.11215	2.07026	2.00000,1.09060,1.72815
27	2.12144	2.06776	2.00000,1.08923,1.69523
45	2.12255	2.06574	2.00000,1.12486,1.60610
5	2.12496	2.06561	2.00000,1.08430,1.68853
16	2.13060	2.06357	2.00000,1.09570,1.63880
37	2.13535	2.05891	2.00000,1.07071,1.66691
4	2.13668	2.05679	2.00000,1.06275,1.67493
11	2.14161	2.05446	2.00000,1.08149,1.60760
1	2.14409	2.05329	2.00000,1.06947,1.62562
29	2.14896	2.04924	2.00000,1.06158,1.61749
41	2.15409	2.04418	2.00000,1.05825,1.59419
8	2.15864	2.03985	2.00000,1.03230,1.63439
12	2.16535	2.03278	2.00000,1.01660,1.63524
38	2.16806	2.02989	2.00000,1.03814,1.56035
9	2.16811	2.02809	2.00000,1.00448,1.64592
47	2.17207	2.02641	2.00000,1.03030,1.55934
33	2.17322	2.0243	2.00000,1.00000,1.63307
43	2.17661	2.02382	2.00000,1.01148,1.59085
20	2.17782	2.02259	2.00000,1.00865,1.59162
34	2.18344	2.01690	2.00000,1.00000,1.58333
30	2.18669	2.01317	2.00000,1.00000,1.56311
10	2.18827	2.00907	2.00000,1.00420,1.53390
24	2.19063	2.00669	2.00000,1.00000,1.53238
35	2.19205	2.00334	2.00000,1.00000,1.51809
19	2.19384	1.99730	2.00000,1.00000,1.49427
23	2.19436	1.99468	2.00000,1.00000,1.48463
18	2.19459	1.99319	2.00000,1.00000,1.47927
32	2.19463	1.99297	2.00000,1.00000,1.47848
3	2.19507	1.98875	2.00000,1.00000,1.46396
0	2.19520	1.9861	2.00000,1.00000,1.45517
15	2.19524	1.98402	2.00000,1.00000,1.44845

Tabela 4.3: Soluções de Pareto via PL - Figura 4.5

Intervalo $[1,3]$ **para os parâmetros** x**,** y **e** z			
Soluções	AD	HV	(x,y,z)
28	3.73452	4.9316	3.00000,1.00173,2.53715
35	3.81216	4.93121	3.00000,1.00000,2.47146
38	3.83636	4.92995	3.00000,1.00000,2.4497
36	3.87577	4.92671	3.00000,1.00000,2.41305
34	3.90411	4.92339	3.00000,1.00000,2.38566
33	3.91788	4.92144	3.00000,1.00000,2.372
13	3.91934	4.92122	3.00000,1.00000,2.37054
8	3.92754	4.91994	3.00000,1.00000,2.36228
2	3.94405	4.91709	3.00000,1.00000,2.34536
27	3.96725	4.91246	3.00000,1.00000,2.32093
19	3.98332	4.90878	3.00000,1.00000,2.30352
32	3.98541	4.90828	3.00000,1.00000,2.30122
5	4.03918	4.89254	3.00000,1.00000,2.23937
17	4.05841	4.8855	3.00000,1.00000,2.21576
24	4.0613	4.88436	3.00000,1.00000,2.21214
39	4.09131	4.87133	3.00000,1.00000,2.17316
0	4.09489	4.8696	3.00000,1.00000,2.16834
29	4.11629	4.85846	3.00000,1.00000,2.13861
15	4.12488	4.85354	3.00000,1.00000,2.12621
14	4.13085	4.84996	3.00000,1.00000,2.11742
26	4.13469	4.84758	3.00000,1.00000,2.1117
22	4.13491	4.84744	3.00000,1.00000,2.11136
20	4.13532	4.84719	3.00000,1.00000,2.11075
30	4.15041	4.83589	3.00000,1.00133,2.08575
40	4.15669	4.83272	3.00000,1.00000,2.07757
6	4.16652	4.82531	3.00000,1.00000,2.06152
4	4.18115	4.81327	3.00000,1.00000,2.03656
7	4.19765	4.79797	3.00000,1.00000,2.00662
21	4.19918	4.79646	3.00000,1.00000,2.00374
16	4.2252	4.76718	3.00000,1.00000,1.95113
31	4.23601	4.75272	3.00000,1.00000,1.92687
3	4.24242	4.74331	3.00000,1.00000,1.91161
10	4.24255	4.74312	3.00000,1.00000,1.91131
18	4.25995	4.7135	3.00000,1.00000,1.8656
37	4.27704	4.67582	3.00000,1.00000,1.81173
9	4.28275	4.66015	3.00000,1.00000,1.79051
25	4.28373	4.65723	3.00000,1.00000,1.78662
1	4.28485	4.65382	3.00000,1.00000,1.78211
23	4.29142	4.63138	3.00000,1.00000,1.75307
11	4.29297	4.62532	3.00000,1.00000,1.74541
12	4.30785	4.46506	3.00000,1.00000,1.56508

Tabela 4.4: Soluções de Pareto via PL - Figura 4.6

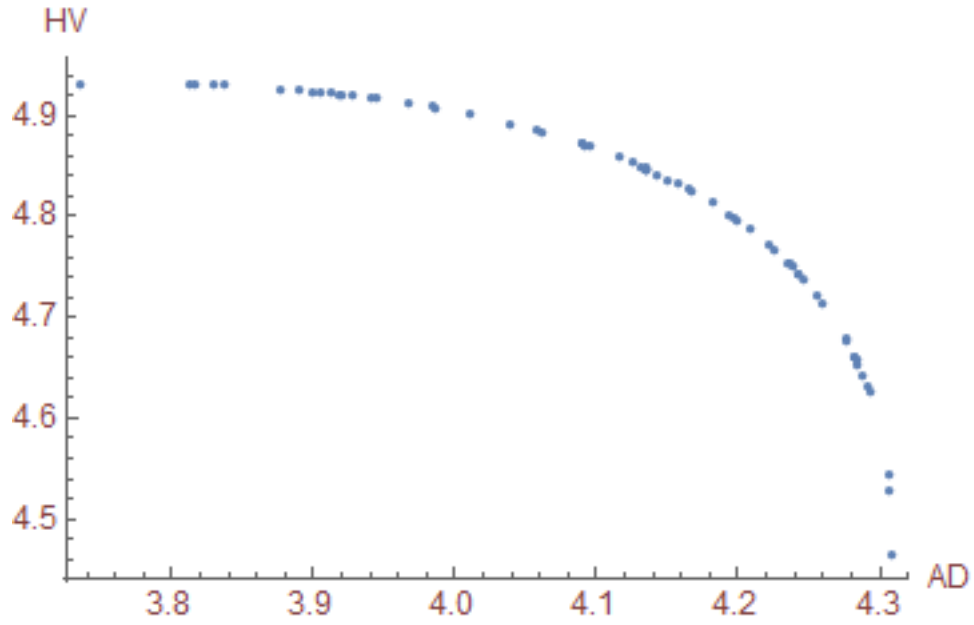

Figura 4.6: Fronteira de Pareto via PL - Tabela 4.4. Fonte própria

4.5 Cenário de testes com melhoria da convergência

Fizemos testes com os modelos HV e AD desenvolvidos por mínimos quadrados neste cenário, observando a Figura 4.7 podemos notar que ela é composta por gráficos em duas colunas. Os gráficos da esquerda representam etapas do algoritmo MOPSO sem convergência, enquanto os gráficos da direita representam as etapas do mesmo algoritmo com convergência. Ao compararmos os gráficos vamos notar que os pontos (em azul) dos gráficos da direita ficaram mais concentrados que os pontos dos gráficos da esquerda. Esta concentração de pontos é decorrente da melhora na convergência do algoritmo MOPSO que propomos neste trabalho. Para rever quais os itens entregues pela implementação computacional e etapas do algoritmo MOPSO vejamos novamente a Figura 4.1 que descreve os itens produzidos pela implementação computacional, espaço de busca, espaço objetivo, soluções não dominadas e a fronteira de Pareto. De maneira geral a convergência faz com que todas as etapas do algoritmo ocorram mais próximo da fronteira de Pareto. Por último podemos notar que a fronteira de Pareto encontrada pelo algoritmo nos dois casos, sem convergência e com convergência, são bem parecidas quanto a forma.

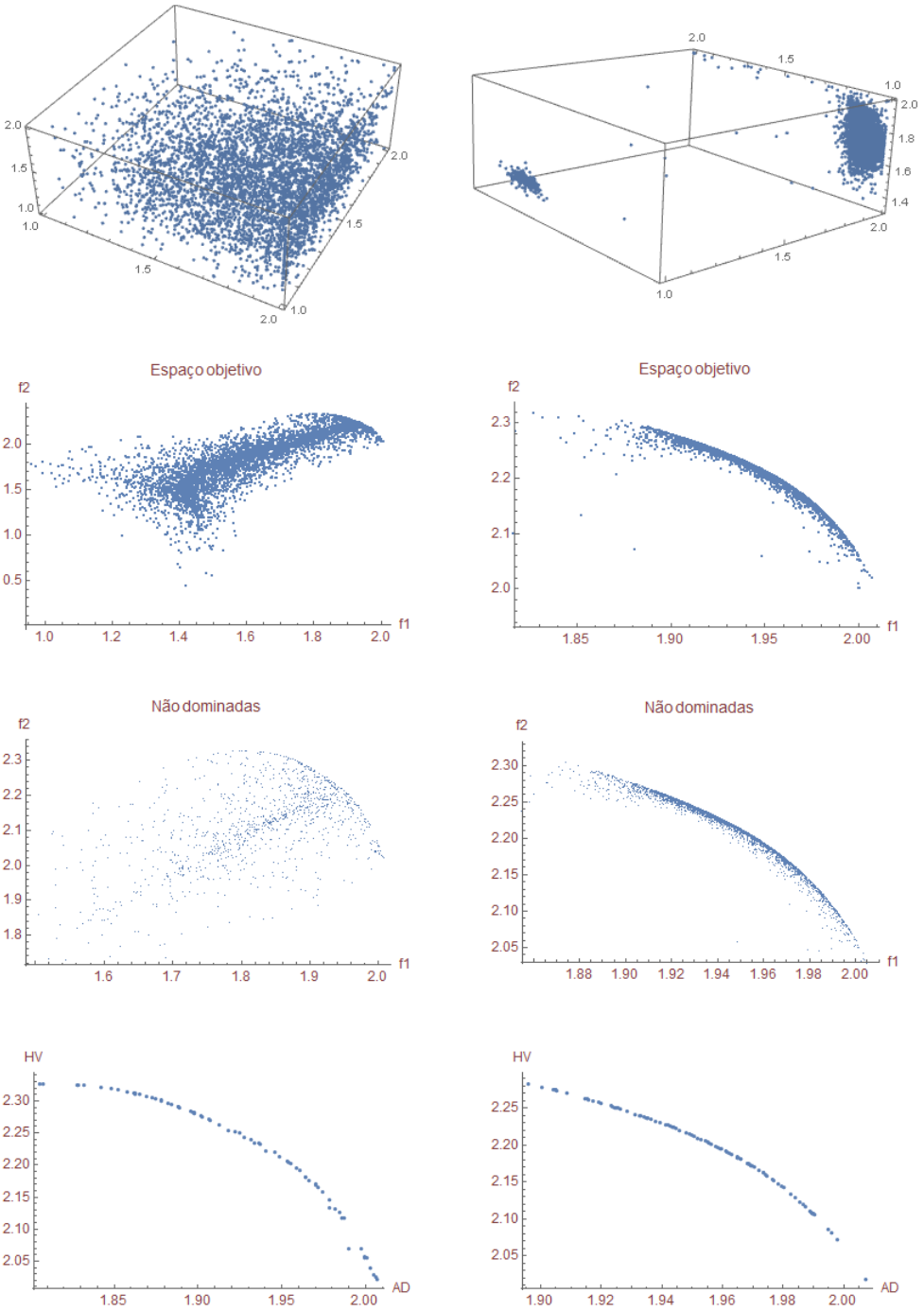

Figura 4.7: Teste visual comparado não convergência × convergência. Fonte própria

Intervalo $[1,2]$ **para os parâmetros de entrada**			
Soluções	HV	AD	(x,y,z)
37	1.81238	2.32706	2.00000,1.00000,1.40701
14	1.81648	2.32675	2.00000,1.00000,1.4162
4	1.82582	2.32545	2.00000,1.00000,1.43747
9	1.82999	2.3246	2.00000,1.00000,1.4471
19	1.84931	2.31827	2.00000,1.00000,1.4931
8	1.85127	2.3174	2.00000,1.00000,1.49788
10	1.85258	2.31679	2.00000,1.00000,1.50109
17	1.85455	2.31583	2.00000,1.00000,1.50596
30	1.85669	2.31474	2.00000,1.00000,1.51128
32	1.86526	2.30978	2.00000,1.00000,1.53286
13	1.8675	2.30832	2.00000,1.00000,1.53858
40	1.86934	2.30707	2.00000,1.00000,1.54333
24	1.87056	2.30622	2.00000,1.00000,1.54647
34	1.87144	2.3056	2.00000,1.00000,1.54874
39	1.87582	2.30145	2.00000,1.00713,1.55437
23	1.88552	2.29255	2.00000,1.01886,1.57117
20	1.90568	2.27279	2.00000,1.03805,1.61416
31	1.90803	2.26868	2.00000,1.05795,1.60862
18	1.90813	2.2685	2.00000,1.00000,1.6499
21	1.91943	2.25702	2.00000,1.05634,1.64409
27	1.92099	2.25525	2.00000,1.05375,1.65045
36	1.92353	2.24688	2.00000,1.10567,1.63447
29	1.93064	2.24138	2.00000,1.09072,1.66316
22	1.93581	2.23142	2.00000,1.03062,1.71278
15	1.93885	2.22941	2.00000,1.10583,1.68584
7	1.94424	2.22009	2.00000,1.12185,1.70119
11	1.95051	2.20809	2.00000,1.05302,1.74961
35	1.95688	2.2	2.00000,1.09925,1.75386
0	1.96153	2.18709	2.00000,1.14797,1.7675
2	1.96689	2.17828	2.00000,1.11885,1.7906
5	1.96724	2.17474	2.00000,1.14783,1.79234
6	1.97085	2.16653	2.00000,1.14418,1.80821
28	1.97139	2.16546	2.00000,1.08943,1.81547
3	1.97393	2.16035	2.00000,1.11272,1.82192
16	1.9742	2.15819	2.00000,1.14063,1.82328
1	1.97784	2.14914	2.00000,1.12059,1.83934
25	1.98697	2.10742	2.00000,1.06167,1.89534
38	1.99884	2.06659	2.00000,1.10301,1.94884
26	2.00132	2.05395	2.00000,1.10566,1.96383
33	2.00515	2.03204	2.00000,1.11218,1.98939
12	2.0068	2.0144	2.00000,1.07064,2.00000

Tabela 4.5: Soluções de Pareto via QM

Intervalo $[1,2]$ **para os parâmetros de entrada**			
Soluções	HV	AD	(x,y,z)
13	1.90423	2.27479	2.00000,1.01996,1.62275
3	1.90494	2.27396	2.00000,1.01843,1.62597
22	1.9052	2.2735	2.00000,1.0338,1.61565
23	1.91467	2.26293	2.00000,1.04202,1.63828
35	1.91911	2.25706	2.00000,1.06232,1.63971
14	1.91997	2.2564	2.00000,1.03865,1.65668
25	1.92308	2.25263	2.00000,1.05697,1.65512
18	1.92373	2.25185	2.00000,1.05262,1.6597
39	1.92476	2.25053	2.00000,1.05425,1.66198
1	1.92549	2.24948	2.00000,1.0494,1.66719
28	1.93257	2.24004	2.00000,1.06832,1.67963
24	1.93395	2.23811	2.00000,1.07115,1.68277
11	1.93478	2.23691	2.00000,1.07322,1.68456
38	1.93777	2.23242	2.00000,1.06771,1.69732
19	1.93932	2.23009	2.00000,1.08682,1.69403
5	1.94112	2.22746	2.00000,1.07849,1.70375
20	1.9435	2.22362	2.00000,1.07725,1.71259
8	1.94815	2.21602	2.00000,1.09096,1.72379
0	1.95069	2.21155	2.00000,1.08986,1.73346
40	1.95284	2.20763	2.00000,1.09154,1.74088
10	1.95496	2.20374	2.00000,1.10176,1.7458
6	1.95931	2.19512	2.00000,1.10493,1.76194
15	1.96227	2.18878	2.00000,1.10168,1.77448
36	1.96478	2.18327	2.00000,1.11172,1.78272
29	1.96602	2.18042	2.00000,1.11147,1.78792
33	1.96832	2.17468	2.00000,1.1029,1.79912
37	1.96843	2.17428	2.00000,1.12632,1.79668
7	1.97032	2.16982	2.00000,1.12153,1.8052
34	1.9719	2.16584	2.00000,1.11625,1.81256
16	1.97332	2.16203	2.00000,1.1137,1.8191
21	1.97334	2.16187	2.00000,1.12295,1.81859
9	1.97431	2.15881	2.00000,1.13085,1.82305
32	1.97657	2.15281	2.00000,1.12329,1.8334
12	1.97693	2.15181	2.00000,1.12238,1.83505
30	1.9785	2.14715	2.00000,1.11571,1.8426
27	1.98007	2.14227	2.00000,1.11573,1.85003
31	1.98265	2.13345	2.00000,1.1311,1.86322
17	1.98412	2.12875	2.00000,1.11192,1.8698
26	1.98971	2.10775	2.00000,1.12834,1.89977
4	1.99781	2.07206	2.00000,1.10725,1.94296
2	2.00713	2.01877	2.00000,1.09349,2.00000

Tabela 4.6: Soluções de Pareto via QM convergente

4.6 Comparação entre os resultados

Não medimos o tempo que o algoritmo MOPSO gastou para produzir uma aproximação da fronteira de Pareto com 70 pontos por exemplo, pois a solução computacional gera dados relativos a etapas intermediárias do algoritmo e não medimos o tempo gasto para que o WM encontre 100 pontos da fronteira de Pareto. O WM foi executado em uma

máquina e o MOPSO foi executado em outra máquina. Para realizarmos uma análise rigorosa dos algoritmos que estão sendo comparados não conhecemos o algoritmo exato usado pelo software Wolfram Mathematica, sendo este tema análise de algoritmos um tema vasto e importante em ciência da computação, mas o tema está fora do escopo deste trabalho. De maneira geral os tempos de execução dos algoritmos ficaram abaixo de 10s.

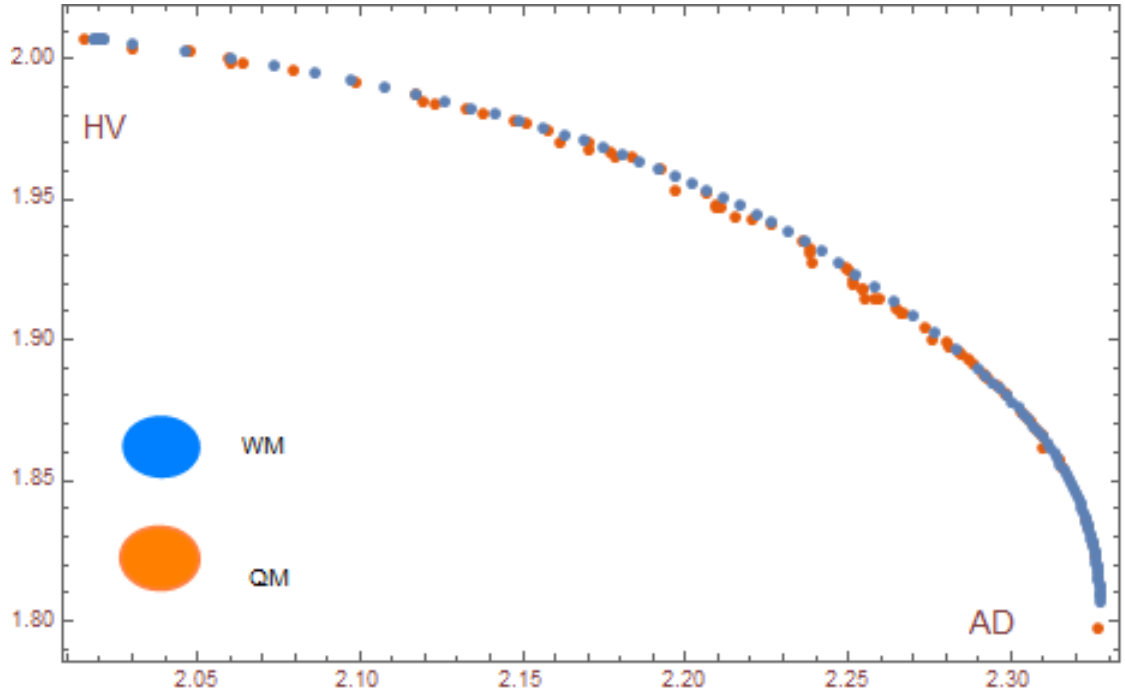

Figura 4.8: Fronteira de Pareto QM-MPSO × WM com escalarização. Fonte própria

Nesta seção estamos comparando resultados de otimização pelo software Wolfram Mathematica com os resultados de otimização pelo MOPSO, os resultados estão em forma de tabela. A seguinte função agregada foi otimizada pelo Wolfram Mathematica, $WM = wAD(x, y, z) + (1 - w)HV(x, y, z)$ com $0 < w < 1$ enquanto os modelos HV e AD foram otimizados pelo MOPSO , conforme o teorema 7 as soluções ótimas do problema escalarizado (agregado) são também soluções eficientes. Conforme a documentação do WM, o comando *Maximize[{f,cons}]* é um otimizador global que sempre encontra um ótimo global se a função f e as restrições forem lineares ou polinomiais. Na tabela correspondente Tabela 4.7, colocamos os valores dos pesos w. Agora vamos dar uma olhada na linha 36 da Tabela 4.7 e a linha 2 da Tabela 4.6, veja que os valores de x, y, z são bem próximos a $(x = 2, y = 1.09, z = 2)$ em ambas tabelas e consequentemente os valores de AD e HV nas duas tabelas, $(AD = 2.0191; HV = 2.007)$ e $(AD = 2.0187; HV = 2.007)$ devem ficar próximos pois foram calculados pelas mesmas funções de (x, y, z).

Vamos comparar a linha 12 da Tabela 4.7 com a linha 10 da Tabela 4.5, observemos que os valores de $(x = 2, y = 1, z = 1.50264)$ e $(x = 2, y = 1, z = 1.501094)$ ficaram bem próximos e consequentemente os valores de AD e HV. Podemos observar quando

comparamos os dados das Tabelas 4.6 e 4.5 com os dados da Tabela 4.7 a aproximação visual da fronteira de Pareto ficou próxima em ambas as abordagens QM-MOPSO e a abordagem da otimização da função escalarizada dos modelos QM pelo *Wolfram*, veja a Figura 4.8 e Figura 4.9. Vamos comparar a linha 35 da Tabela 4.3 com a linha 24 da Tabela 4.8, podemos notar que as diferenças foram pequenas e que esta tendência se confirma quando comparamos os dados da Tabela 4.3 gerados pelos modelos PL com os dados da Tabela 4.8 obtida através do software *Wolfram Mathematica*, verifique a Figura 4.10.

O que fizemos aqui foi comparar pontualmente alguns resultados do algoritmo MOPSO aplicados aos modelos ajustados por QM com os resultados do *Wolfram* na maximização da respectiva função agregada. Repetimos este teste agora comparando alguns resultados do algoritmo MOPSO aplicados aos modelos ajustados pela abordagem PL com os resultados obtidos pela maximização da respectiva função agregada pelo *Wolfram*. De forma geral quando plotamos os dados das tabelas que aproximam a fronteira de Pareto tivemos um bom resultado, Figura 4.8 e Figura 4.10.

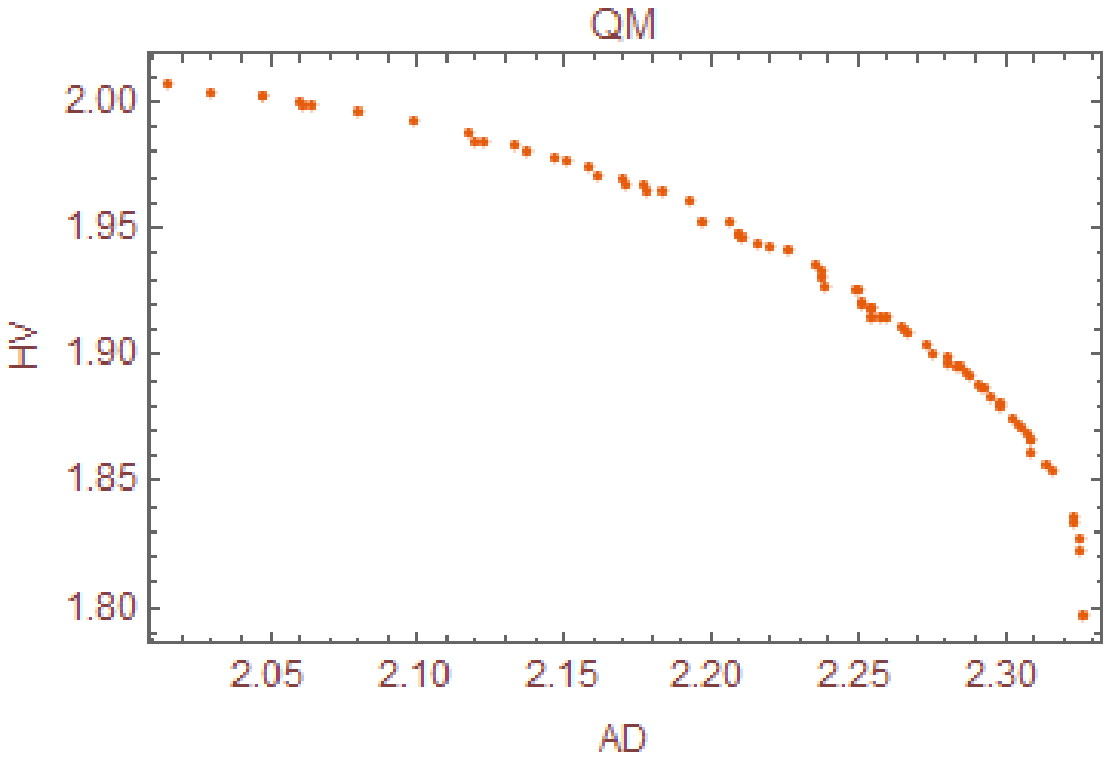

Figura 4.9: Fronteira de Pareto por QM e MOPSO. Fonte própria

WM - Intervalo $[1,2]$ **para os parâmetros** x**,** y **e** z			
Soluções	AD	HV	(x, y, z, w)
38	2.01865	2.00713	2.00000,1.09287,2.00000,0.975
37	2.01919	2.00711	2.00000,1.09572,2.00000,0.95
36	2.01975	2.00707	2.00000,1.09872,2.00000,0.925
35	2.02033	2.00702	2.00000,1.10186,2.00000,0.9
34	2.02094	2.00694	2.00000,1.10514,2.00000,0.875
33	2.04597	2.00279	2.00000,1.1105,1.97365,0.85
32	2.07986	1.99624	2.00000,1.11527,1.93454,0.825
31	2.10725	1.98993	2.00000,1.11801,1.89973,0.8
30	2.12991	1.98383	2.00000,1.11892,1.86829,0.775
29	2.14909	1.97786	2.00000,1.11811,1.83951,0.75
28	2.16571	1.97195	2.00000,1.11566,1.81284,0.725
27	2.1805	1.96599	2.00000,1.11157,1.78776,0.7
26	2.19401	1.95985	2.00000,1.10579,1.76386,0.675
25	2.20673	1.95337	2.00000,1.09822,1.74076,0.65
24	2.21906	1.94635	2.00000,1.0887,1.7181,0.625
23	2.23142	1.93853	2.00000,1.07699,1.69551,0.6
22	2.2442	1.92955	2.00000,1.06277,1.67262,0.575
21	2.25785	1.91892	2.00000,1.0456,1.64899,0.55
20	2.27292	1.90594	2.00000,1.02486,1.62412,0.525
19	2.28999	1.88968	2.00000,1.00000,1.59747,0.5
18	2.29555	1.88384	2.00000,1.00000,1.58158,0.475
17	2.30038	1.87823	2.00000,1.00000,1.56655,0.45
16	2.30458	1.87283	2.00000,1.00000,1.55237,0.425
15	2.30823	1.86764	2.00000,1.00000,1.53894,0.4
14	2.3114	1.86264	2.00000,1.00000,1.5262,0.375
13	2.31413	1.85783	2.00000,1.00000,1.51412,0.35
12	2.31649	1.85321	2.00000,1.00000,1.50264,0.325
11	2.31852	1.84875	2.00000,1.00000,1.49171,0.3
10	2.32026	1.84445	2.00000,1.00000,1.4813,0.275
9	2.32173	1.8403	2.00000,1.00000,1.47135,0.25
8	2.32298	1.83629	2.00000,1.00000,1.46187,0.225
7	2.32402	1.83244	2.00000,1.00000,1.4528,0.2
6	2.32489	1.82871	2.00000,1.00000,1.44413,0.175
5	2.32559	1.8251	2.00000,1.00000,1.43582,0.15
4	2.32614	1.82162	2.00000,1.00000,1.42785,0.125
3	2.32657	1.81825	2.00000,1.00000,1.4202,0.1
2	2.32688	1.81499	2.00000,1.00000,1.41286,0.075
1	2.32709	1.81184	2.00000,1.00000,1.4058,0.05
0	2.32721	1.80878	2.00000,1.00000,1.39902,0.025

Tabela 4.7: Soluções de Pareto *Wolfram Mathematica* e QM para o intervalo $[1, 2]$

WM - Intervalo $[1,2]$ **para os parâmetros** x**,** y **e** z			
Soluções	AD	HV	(x, y, z, w)
38	2.18836	2.01077	2.00000,1.00000,1.55118,0.39
37	2.18874	2.01017	2.00000,1.00000,1.54831,0.38
36	2.1891	2.00956	2.00000,1.00000,1.54546,0.37
35	2.18945	2.00895	2.00000,1.00000,1.54261,0.36
34	2.18979	2.00834	2.00000,1.00000,1.53978,0.35
33	2.19012	2.00771	2.00000,1.00000,1.53695,0.34
32	2.19044	2.00709	2.00000,1.00000,1.53414,0.33
31	2.19074	2.00645	2.00000,1.00000,1.53134,0.32
30	2.19104	2.00581	2.00000,1.00000,1.52856,0.31
29	2.19132	2.00517	2.00000,1.00000,1.52578,0.3
28	2.19159	2.00452	2.00000,1.00000,1.52302,0.29
27	2.19185	2.00386	2.00000,1.00000,1.52026,0.28
26	2.1921	2.0032	2.00000,1.00000,1.51752,0.27
25	2.19234	2.00254	2.00000,1.00000,1.51479,0.26
24	2.19257	2.00187	2.00000,1.00000,1.51207,0.25
23	2.19279	2.00119	2.00000,1.00000,1.50936,0.24
22	2.193	2.00051	2.00000,1.00000,1.50667,0.23
21	2.1932	1.99983	2.00000,1.00000,1.50398,0.22
20	2.19339	1.99914	2.00000,1.00000,1.50129,0.21
19	2.19357	1.99845	2.00000,1.00000,1.49864,0.2
18	2.19373	1.99775	2.00000,1.00000,1.49599,0.19
17	2.19389	1.99705	2.00000,1.00000,1.49334,0.18
16	2.19404	1.99634	2.00000,1.00000,1.49072,0.17
15	2.19419	1.99563	2.00000,1.00000,1.48809,0.16
14	2.19432	1.99491	2.00000,1.00000,1.48547,0.15
13	2.19444	1.9942	2.00000,1.00000,1.48288,0.14
12	2.19455	1.99347	2.00000,1.00000,1.48029,0.13
11	2.19466	1.99275	2.00000,1.00000,1.4777,0.12
10	2.19475	1.99201	2.00000,1.00000,1.47513,0.11
9	2.19484	1.99128	2.00000,1.00000,1.47257,0.1
8	2.19491	1.99054	2.00000,1.00000,1.47002,0.09
7	2.19498	1.9898	2.00000,1.00000,1.46748,0.08
6	2.19504	1.98905	2.00000,1.00000,1.46495,0.07
5	2.1951	1.9883	2.00000,1.00000,1.46243,0.06
4	2.19514	1.98755	2.00000,1.00000,1.45992,0.05
3	2.19518	1.98679	2.00000,1.00000,1.45741,0.04
2	2.1952	1.98603	2.00000,1.00000,1.45492,0.03
1	2.19522	1.98526	2.00000,1.00000,1.45245,0.02
0	2.19523	1.9845	2.00000,1.00000,1.44998,0.01

Tabela 4.8: Soluções de Pareto *Wolfram Mathematica* e PL para o intervalo $[1,2]$

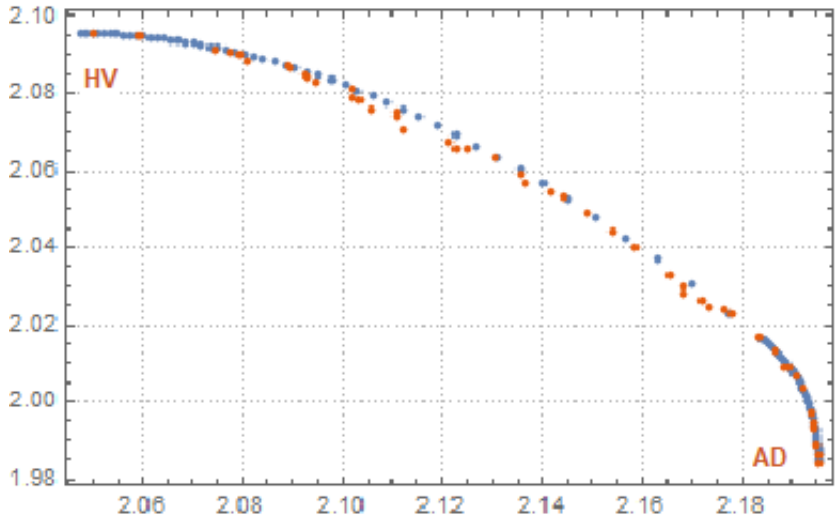

Figura 4.10: Fronteira de Pareto PL-MOPSO × WM com escalaricação. Fonte própria

4.7 Conclusões dos resultados computacionais

Quanto aos modelos matemáticos preditivos eles conseguem uma boa aproximação da fronteira de Pareto, tivemos poucas varições nas medidas da fronteira de Pareto, ou seja, a diferença entre valores mínimos entre os modelos AD ficou próximo de 0.09 e a diferença entre os valores mínimos entre os modelos HV ficou próximo de 0.03 e a diferença entre os valores máximos dos modelos HV ficou próximo de 0.08 e a diferença entre os valores máximos dos modelos AD ficou próximo de 0.9.

Os resultados por MOPSO de ambos os modelos quando comparados com os resultados do Wolfram indicam que o algoritmo MOPSO conseguiu resultados semelhantes aos resultados oferecidos pelo Wolfram. Sendo o software Wolfram Mathematica usado nestas comparações por ser uma opção de comparação.

Ao compararmos os modelos pela abordagem PL versus abordagem QM podemos observar que os modelos pela abordagem PL sugerem valores mais altos para AD e HV, pensando em maximização conforme proposto no problema inicial podemos dizer que a previsão dos modelos PL são melhores, pois os mesmos fizeram uma previsão de valores mais altos para AD e HV. Agora qual modelo melhor descreve os experimentos é uma questão para ser verificada em laboratório. De certa forma podemos ter uma noção conjunta dos limites de valores ótimos dos modelos olhando para o ponto ideal dos modelos, (AD_{max}, HV_{max}). Para o modelo PL temos o seguinte ponto ideal $(2.19524, 2.09555)$. Na linha 16 da Tabela 4.3 encontramos $(2.130604, 2.063569)$ um valor razoável.

Capítulo 5

Conclusões e Perspectivas Futuras

Neste capítulo final resumimos as questões principais abordados nesta dissertação apresentando na seção 5.1 as conclusões obtidas por meio deste trabalho, e na seção 5.2 apresentamos sugestões para trabalhos futuros.

5.1 Conclusões

Inicialmente quando lemos o artigo de Rafieerad et al. (2017), identificamos a oportunidade de propormos uma metodologia que melhora a condição experimental no desenvolvimento de materiais avançados (biomateriais), nossa metodologia introduz formas de ajustes de coeficientes mais adequadas. Em suma nossa metodologia é uma resposta positiva ao cenário ineficiente descrito no artigo Rafieerad et al. (2017). Pela metodologia obtemos computacionalmente os melhores valores para os dados e sugerimos o teste em laboratório apenas destes melhores dados. No desenvolvimento de biomateriais há situações onde é requerido o ajuste simultâneo de duas propriedades do material. Evitamos a situação de tentativa e erro nos experimentos propondo modelos matemáticos preditivos para a força de adesão AD e a dureza HV em seguida fizemos a otimização computacional destes modelos por meio do algoritmo heurístico multiobjetivo conhecido como algoritmo baseado em inteligência coletiva, enxame de partículas, nossos modelos foram propostos baseados em dados experimentais existentes no trabalho de Rafieerad et al. (2017), poucos dados. Como resultado da otimização computacional multiobjetivo temos uma previsão dos possíveis parâmetros ótimos para que o material seja ajustado em laboratório e se caso estes parâmetros não forem bons, não derem os resultados esperados em laboratório, nossa metodologia melhora o ajuste dos coeficientes dos modelos recebendo mais dados experimentais, dados preditivos que foram testados e não se comportaram como previsto pelos modelos, para recapitularmos sobre a metodologia podemos observar a Figura 3.1 do Capítulo 3.

Sobre os modelos, trabalhamos com dois modelos diferentes tanto para HV quanto para AD, os coeficientes dos modelos foram calculados por estratégias de ajustes dife-

rentes, baseado em uma abordagem por programação linear e na técnica de quadrados mínimos. E por ultimo implementamos um algoritmo multiobjetivo (MOPSO) baseado em enxame de partículas para otimizar os modelos. Melhoramos a convergência do algoritmo fazendo ajustes na equação do deslocamento das partículas, dando então nossa contribuição ao tema computação bio inspirada. Também fizemos outro ajuste na equação do movimento das partículas, de forma que soluções não dominadas sejam capturadas do espaço de busca e não do repositório.

Baseado nos testes, nossos modelos foram coerentes quando otimizados por MOPSO e pelo WM, os resultados da aproximação da curva de Pareto ficaram muito próximos tanto por MOPSO e WM, caso necessário rever este ponto basta verificar a Figura 4.8 do capítulo 4, este bom resultado pode ser atribuído ao algoritmo heurístico usado para as otimizações, era de se esperar que os modelos $HV(x, y, z)$ e $AD(x, y, z)$ tivessem valor zero quando $x = 0, y = 0, z = 0$ simultaneamente, logo incluímos este ponto aos dados experimentais apenas durante o ajuste por quadrados mínimos, pois era necessário mais um ponto experimental para evitar a situação de indeterminação durante o ajuste por quadrados mínimos. Não fazendo sentido uma situação onde $x = 0, y = 0, z = 0$ com $HV \neq 0$ e $AD \neq 0$.

Um ponto importante que conseguimos verificar foi em relação ao algoritmo MOPSO, em específico durante a varredura do espaço de buscas quando uma partícula pesquisa uma nova posição, existem várias novas posições neste espaço que domina a posição atual, o leitor poderá observar a Figura 3.5 no Capítulo 3 e notar que todas as posições acima da posição $A(f_1, f_2)$ tem valores melhores que os da posição A, consequentemente se a partícula se mover em uma próxima iteração para a região em amarelo irá atingir posições que dominam a posição atual e para uma estimativa numérica desta dificuldade ou facilidade basta dividirmos a área compreendida entre os eixos f_1 e f_2 com A na origem dos eixos, pela área total do espaço objetivo, nos testes conseguimos constatar que a direção do vetor $\overrightarrow{g}$ pode ser escolhida aleatoriamente que o algoritmo convergirá para a fronteira de Pareto, este comportamento se justificou devido a quantidade de posições que dominam uma posição atual no espaço objetivo, logo o papel do repositório foi apenas de registrar as posições que tendem a fronteira de Pareto. Conforme mostramos se a direção de $\overrightarrow{g}$ for escolhida de tal forma que fique entre os gradientes das funções objetivo, o algoritmo convergirá rapidamente para a fronteira de Pareto, fazendo com que as partículas se movimentem no espaço de busca sempre para novas posições que domine a posição atual.

Quanto ao conceito de repositório proposto nos algoritmos MOPSO foi possível analisar seu papel e fazer suposições sobre sua usabilidade, neste problema verificamos que há alternativas melhores que eleger membros do repositório como líder, ou seja, posições armazenadas no repositório para compor o vetor $\overrightarrow{g}$.

A metodologia sugerida neste trabalho tem aplicação na área de materiais e em qual-

quer área do conhecimento que forneça alguma quantidade de dados experimentais e necessite de novos dados gerados computacionalmente por modelos matemáticos para serem testados em laboratório.

Não comparamos pontos da fronteira de Pareto que obtemos, com os pontos da fronteira de Pareto fornecidos pelo artigo de Rafieerad et al. (2017), pois no artigo não são mencionadas as coordenadas dos pontos, em nossa trabalho apresentamos em tabelas as coordenadas dos pontos da fronteira de Pareto e os respectivos valores das funções objetivo. No artigo o autor ajustou os coeficientes dos modelos usando um algoritmo heurístico referenciado no artigo como Smart PSO, em nosso trabalho optamos por ajustar os coeficientes dos modelos por métodos determinísticos, uma abordagem PL e por mínimos quadrados. Em termos de comparação de resultados optamos por comparar nossos resultados de otimização multiobjetivo com resultados obtidos pelo software Wolfram Mathematica onde nossos resultados ficaram próximos dos resultados do Wolfram Mathematica. Não comparamos nossos modelos com os do artigo pois apesar dos dados para ajustes serem os mesmos, os modelos foram obtidos por abordagens diferentes.

5.2 Perspectivas futuras

Para os próximos trabalhos, é indicado um estudo probabilístico do movimento das partículas, uma verificação do comportamento do algoritmo MOPSO no melhor caso e no pior caso, neste trabalho fizemos uma breve análise quantitativa e qualitativa da dominância das posições das partículas quando se deslocam, na verdade fizemos uma breve análise para uma única partícula, um estudo para duas ou mais partículas permite um entendimento mais aprofundo deste ponto do algoritmo (caminhar em direções melhores), gerando resultados numéricos sobre padrões e comportamento do algoritmo MOPSO.

Neste trabalho a parte principal da implementação computacional foi a do algoritmo heurístico MOPSO, ou seja, uma implementação multiobjetivo; como sugestão complementar, uma implementação secundário e mono-objetivo para o mesmo problema, agora com o algoritmo PSO e com base na função $G(\overrightarrow{x})$ de cálculo de dominância da Seção 2.3.1, esta abordagem tem a vantagem de maior simplicidade.

Além destas sugestões um próximo trabalho poderia ser complementado com a criação de posições artificiais no repositório onde é possível controlar melhor novas referências de posições que entram no repositório e melhorar o desempenho direcionando as buscas para regiões mais apropriadas.

Neste trabalho não implementamos o smart PSO que calcula os coeficientes dos modelos, mencionado no trabalho de Rafieerad et al. (2017), usamos a abordagem PL que também minimiza a somatória dos desvios absolutos. Como trabalho futuro podemos inserir pequenas perturbações nos coeficientes dos modelos que propomos e verificar e comparar as soluções não dominadas obtidas antes de depois das pertubações. Pode-

mos também ajustar os coeficientes dos modelos propostos no artigo de forma heurística, usando o algoritmo Smart PSO, como o algoritmo é heurístico vamos obter coeficientes diferentes cada vez que o algoritmo executa, esta estratégia se aplica para testes dos modelos propostos no artigo, onde podemos comparar as soluções não dominadas geradas pela otimização de cada modelo obtido.

Referências Bibliográficas

Abensur, E. O. (2012). Um modelo multiobjetivo de otimização aplicado ao processo de orçamento de capital. *Gestão & Produção*, 19(4):747–758.

Anton, H. and Rorres, C. (2010). *Elementary linear algebra: applications version*. John Wiley & Sons.

Arenales, M., Morabito, R., Armentano, V., and Yanasse, H. (2015). *Pesquisa operacional: para cursos de engenharia*. Elsevier Brasil.

Arroyo, J. E. C., dos Santos Ottoni, R., and de Paiva Oliveira, A. (2011). Multi-objective variable neighborhood search algorithms for a single machine scheduling problem with distinct due windows. *Electronic Notes in Theoretical Computer Science*, 281:5–19.

Bector, C. R. (1973). Duality in nonlinear fractional programming. *Zeitschrift für Operations Research*, 17:183–193.

Chankong, V. and Haimes, Y. Y. (1983). *Multiobjective Decision Making: Theory and Methodology*. North-Holland, Elsevier Science Publishing Co., Inc., New York. (North Holland series in system science and engineering; 8).

Cohon, J. L. (1978). *Multiobjective programming and planning*. Academic Press, New York.

Collette, Y. and Siarry, P. (2013). *Multiobjective optimization: principles and case studies*. Springer Science & Business Media.

Cramer, N. L. (1985). A representation for the adaptive generation of simple sequential programs. In *Proceedings of the first international conference on genetic algorithms*, pages 183–187.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Czyżak, P. and Jaszkiewicz, A. (1998). Pareto simulated annealing a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7:34–47.

Faradonbeh, R. S., Armaghani, D. J., Monjezi, M., and Mohamad, E. T. (2016). Genetic programming and gene expression programming for flyrock assessment due to mine blasting. *International Journal of Rock Mechanics and Mining Sciences*, 88:254–264.

Faro, C. and Lachtermacher, G. (2012). *Introdução à Matemática Financeira*.

Friedlander, A. (2012). *Elementos de programação não-linear*. Notas de Aulas.

Gandibleux, X. and Ehrgott, M. (2006). *Multiple criteria optimization: state of the art annotated bibliographic surveys*, volume 52. Springer Science & Business Media.

Gendreau, M. and Potvin, J.-Y. (2010). *Handbook of metaheuristics*, volume 2. Springer.

Gene, F. C. (2001). expression programming: A new adaptive algorithm for solving problems [j]. *Complex Systems*, 13(2):87–129.

Guignard, M. (1969). Generalized Kuhn-Tucker conditions for mathematical programming problems in a Banach space. *SIAM Journal on Control*, 7:232–241.

Hillier, F. S. (2015). *Introduction to operations research*. Tata McGraw-Hill Education.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks IV*, volume 1000.

Leong, W. F. (2008). *Multiobjective Particle Swarm Optimization: integration of dynamic population and multiple-swarm concepts and constraint handling*. PhD thesis, Oklahoma State University.

Liang, Z. A., Huang, H. X., and Pardalos, P. M. (2001). Optimality conditions and duality for a class of nonlinear fractional programming problems. *Journal of Optimization Theory and Applications*, 110:611–619.

Lin, Q., Li, J., Du, Z., Chen, J., and Ming, Z. (2015). A novel multi-objective particle swarm optimization with multiple search strategies. *European Journal of Operational Research*, 247(3):732–744.

Marler, R. T. and Arora, J. S. (2004). Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395.

McCallum, W. G., Hughes-Hallett, D., Osgood, B. G., and Flath, D. (2005). *Calculus: multivariable calculus*. Wiley.

Miettinen, K. (2012). *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media.

Miettinen, K. M. (1999). *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, International Series in Operations Research & Management Science, Boston.

Myers, R. H., Montgomery, D. C., and Anderson-Cook, C. M. (2009). *Response Surface Methodology: Process and Product Optimization Using Designed Experiments (Wiley Series in Probability and Statistics)*. Wiley New York.

Odu, G. O. and Charles-Owaba, O. E. (2013). Review of multi-criteria optimization methods–theory and applications. *IOSRJEN*, 3:1–14.

Oliveira, W. A. (2011). *Condições de Otimalidade em Programação Multiobjetivo Fracional Quadrático*. PhD thesis.

Osuna-Gómez, R., Jiménez, B. H., and Salles Neto, L. L. (2010). Duality theory for the multiobjective nonlinear programming involving generalized convex functions. In Arana-Jimenez, M., Rufián-Lizana, A., and Ruiz-Garzón, G., editors, *Optimality Conditions in Vector Optimization*, volume 1, pages 105–118. Sharjah: Bentham, 1 ed.

Osuna-Gómez, R., Rufián-Lizana, A., and Canales, P. R. (2000). Multiobjective fractional programming with generalized convexity. *Sociedad de Estadística e Investigación Operativa, TOP*, 8(1):97–110.

Pareto, V. (1896). *Cours d'Economie Politique*. Rouge, Lausanne, Switzerland.

Pasandideh, S. H. R., Niaki, S. T. A., and Sharafzadeh, S. (2013). Optimizing a bi-objective multi-product epq model with defective items, rework and limited orders: Nsga-ii and mopso algorithms. *Journal of Manufacturing Systems*, 32(4):764–770.

Rafieerad, A., Bushroa, A., Nasiri-Tabrizi, B., Fallahpour, A., Vadivelu, J., Musa, S., and Kaboli, S. (2016). Gep-based method to formulate adhesion strength and hardness of nb pvd coated on ti–6al–7nb aimed at developing mixed oxide nanotubular arrays. *Journal of the mechanical behavior of biomedical materials*, 61:182–196.

Rafieerad, A., Bushroa, A., Nasiri-Tabrizi, B., Kaboli, S., Khanahmadi, S., Amiri, A., Vadivelu, J., Yusof, F., Basirun, W., and Wasa, K. (2017). Toward improved mechanical, tribological, corrosion and in-vitro bioactivity properties of mixed oxide nanotubes on ti–6al–7nb implant using multi-objective pso. *Journal of the mechanical behavior of biomedical materials*, 69:1–18.

Romero, C. (1993). *Teoría de la Decisión Multicriterio: Conceptos, Técnicas y Aplicaciones*. Alianza Universidad Textos, Madrid, Alianza Editorial. ISBN: 9788420681443.

Santos, L. B., Osuna-Gómez, R., and Rojas-Medar, M. A. (2008). Nonsmooth multiobjective fractional programming with generalized convexity. *Resvista Integración, Escuela de Matemáticas Universidad Industrial de Santander*, 26(1):1–12.

Silva, I. d., Spatti, D. H., and Flauzino, R. A. (2010). Redes neurais artificiais para engenharia e ciências aplicadas. *São Paulo: Artliber*, 23(5):33–111.

Stiemke, E. (1915). Über positive lösungen homogener linearer gleichungen. *Mathematische Annalen*, 76(2):340–342.

Taha, H. A. (2007). *Operations Research an Introduction*. Pearson Prentice Hall, New Jersey,USA.

Talbi, E. G. (2009). *Metaheuristics from design to implementation*. Jhon Wiley, New Jersey,USA.

Apêndice A

Implementações do algoritmo do gradiente com adição de heurística

No algoritmo original sempre se anda na direção do gradiente para encontrar um máximo da função, nesta implementação foi introduzido um operador de rotação do R^2 $gi[\theta]$, este operador permite que na implementação do algoritmo escolha-se uma direção arbitrária que forme um angulo θ com o gradiente, ao invés de andar sempre na direção do gradiente. Por meio desta variação heurística do algoritmo é possível verificar que existe uma direção próxima a do gradiente onde o desempenho do algoritmo é melhor. No problema exemplo a direção testada que melhora em cerca de 50% a quantidade de passos do algoritmo é de 12^o graus. Para testes de execução com a mesma direção do gradiente basta escolher o angulo $\theta = 0$ como ilustrado no código $gi[0]$. A função a ser maximizada foi extraído de Taha (2007).

A.1 Código do exemplo

/*EXECUÇÃO DO ALGORITMO NO SOFTWARE WOLFRAM MATHEMATICA.*/

/*DEFINIÇÃO DE FUNÇÕES E VARIÁVEIS*/

gi[θ_]:={{Cos[θDegree], Sin[θDegree]}, {−Sin[θDegree], Cos[θDegree]}};

f[x_,y_]=$4x + 6y - 2x^2 - 2x * y - 2y^2$;

$gd[x\_, y\_] = Grad[f[x, y], \{x, y\}]$

G[x_,y_,r_]={x,y} + gi[0].gd[x,y]*r;

M=G[x,y,r];

```
h[r_]=f[M[[1]],M[[2]]];
U=Solve[D[h[r],r]==0,r];
R[x_,y_] = (13-14x+5x^2-16y+8xy+5y^2)/(19-22x+7x^2-23y+13xy+7y^2);
gd[x_,y_] =Grad[f[x,y],{x,y}];


/*INICIALIZAÇÃO*/
M=G[1,1,R[1,1]]
/*LOOP*/
While[N[Norm[gd[M[[1]],M[[2]]]]] > 10^-3,Print[M = G[M[[1]],M[[2]],R[M[[1]],M[[2]]]]]]
/*SAIDA*/
N[M=G[M[[1]],M[[2]],R[M[[1]],M[[2]]]]]
N[Norm[gd[M[[1]],M[[2]]]]]
```

Método do gradiente do problema exemplo implementado em C com precisão de 10^{-3}

```c
#include <stdio.h>
#include <math.h>

double (*f)(double,double);
double F(double x,double y)
{

   return 4*x +6*y 2*x*x 2*x*y 2*y*y;

}
void  grad(double x,double y,double A[1][2] ){
     A[0][0]=4 4*x 2*y;
     A[0][1]=6 2*x 4*y;


}
void g(double x,double y,double r,double A[1][2])
{
   A[0][0]=x+(4 4*x 2*y)*r;
   A[0][1]=y+(6 2*x 4*y)*r;

}
double R(double x,double y)
{
return
(double)
((13    14*x + 5*x*x    16*y + 8*x*y + 5*y*y)/(4*(19    22*x + 7*x*x    23*y + 13*x*y + 7*y*y)));
}
int main() {
    int i=0;
    double MA[1][2];
```

```
32      double G[1][2];
33      f=&F;
34      g(1.0,1.0,R(1.0,1.0),G);
35      printf("␣X%d=(%f,␣%f)␣\n",++i,G[0][0],G[0][1]);
36      g(G[0][0],G[0][1],R(G[0][0],G[0][1]),G);
37      printf("␣X%d=(%f,␣%f)␣\n",++i,G[0][0],G[0][1]);
38
39      for(;;){
40        g(G[0][0],G[0][1],R(G[0][0],G[0][1]),G);
41
42        printf("␣X%d=(%f,␣%f)␣\n",++i,G[0][0],G[0][1]);
43
44        grad(G[0][0],G[0][1],MA);
45
46        if(sqrt(MA[0][0]*MA[0][0]+MA[0][1]*MA[0][1])<=0.001)
47         {
48            break;
49         }
50  }
51
52  }
```

```
1   OUTPUT
2   X1=(0.500000, 1.000000)
3   X2=(0.500000, 1.250000)
4   X3=(0.375000, 1.250000)
5   X4=(0.375000, 1.312500)
6   X5=(0.343750, 1.312500)
7   X6=(0.343750, 1.328125)
8   X7=(0.335938, 1.328125)
9   X8=(0.335938, 1.332031)
10  X9=(0.333984, 1.332031)
11  X10=(0.333984, 1.333008)
12  X11=(0.333496, 1.333008)
```

Apêndice B

Cálculo dos coeficientes do modelo para força de adesão e dureza

Calculamos os coeficientes da força de adesão AD por programação linear, usando o software GUSEK. O modelo foi implementado em linguagem AMPL.

B.1 Código do modelo por programação linear para a força de adesão

```
var x0;
var x1;
var x2;
var x3;
var x4;
var x5;
var x6;
var x7;
var x8;
var x9;
var E1>=0;
var e1>=0;
var E2>=0;
var e2>=0;
var E3>=0;
var e3>=0;
var E4>=0;
var e4>=0;
var E5>=0;
var e5>=0;
var E6>=0;
var e6>=0;
var E7>=0;
var e7>=0;
var E8>=0;
var e8>=0;
var E9>=0;
var e9>=0;

```

```
30
31 minimize z:E1+e1+E2+e2+E3+e3+E4+e4+E5+e5+E6+e6+E7+e7+E8+e8+E9+e9;
32
33
34
35 s.t. r1:x0 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9+E1 e1=1.090;
36 s.t. r2:x0+x1+1.5*x2+1.5*x3+1.5*x4+1.5*x5+2.25*x6 + x7+2.25*x8+2.25*x9+E2 e2=1.261;
37 s.t. r3:x0 + x1 + 2*x2 + 2*x3 + 2*x4 + 2*x5 + 4*x6 + x7 + 4*x8 + 4*x9+E3 e3=1.562;
38 s.t. r4:x0 + 1.5*x1+x2+1.5*x3+1.5*x4+2.25*x5+1.5*x6+2.25*x7+x8+2.25*x9+E4 e4=1.529;
39 s.t. r5:x0 + 1.5*x1+1.5*x2+2*x3+2.25*x4+3*x5+3*x6+2.25*x7+2.25*x8+4*x9+E5 e5=1.431;
40 s.t. r6:x0 + 1.5*x1+2*x2+x3+3*x4+1.5*x5+2*x6+2.25*x7+4*x8+x9+E6 e6=1.000;
41 s.t. r7:x0 + 2*x1 + x2 + 2*x3 + 2*x4 + 4*x5 + 2*x6 + 4*x7 + x8 + 4*x9+E7 e7=2.000;
42 s.t. r8:x0 + 2*x1+1.5*x2+x3+3*x4+2*x5+1.5*x6+4*x7+2.25*x8+x9+E8 e8=1.623;
43 s.t. r9:x0 + 2*x1+2*x2+1.5*x3+4*x4+3*x5+3*x6+4*x7+4*x8+2.25*x9+E9 e9=1.755;
44
45 /*s.t. r10:(E1 e1+E2 e2+E3 e3+E4 e4+E5 e5+E6 e6+E7 e7+E8 e8+E9 e9)=0.246074;*/
46
47  end;
```

B.2 Resultado da execução no Gusek do modelo por programação linear para o cálculo dos coeficientes para o modelo da força de adesão

```
1  Problem:    min
2  Rows:       10
3  Columns:    28
4  Non zeros:  126
5  Status:     OPTIMAL
6  Objective:  z = 0 (MINimum)
7
8     No.   Row name   St   Activity      Lower bound   Upper bound    Marginal
9
10       1  z          B             0
11       2  r1         NS         1.09          1.09             =       < eps
12       3  r2         NS        1.261         1.261             =       < eps
13       4  r3         NS        1.562         1.562             =       < eps
14       5  r4         NS        1.529         1.529             =       < eps
15       6  r5         NS        1.431         1.431             =       < eps
16       7  r6         NS            1             1             =       < eps
17       8  r7         NS            2             2             =       < eps
18       9  r8         NS        1.623         1.623             =       < eps
19      10  r9         NS        1.755         1.755             =       < eps
20
21    No. Column name  St   Activity      Lower bound   Upper bound    Marginal
22
23       1  x0         B     0.617619
24       2  x1         NF           0                                  < eps
25       3  x2         B         0.422
26       4  x3         B         0.698
27       5  x4         B     0.767238
28       6  x5         B     0.183238
29       7  x6         B     0.787238
30       8  x7         B     0.520381
31       9  x8         B     0.112381
32      10  x9         B     0.639619
```

```
      11 E1          NL          0          0          1
      12 e1          NL          0          0          1
      13 E2          NL          0          0          1
      14 e2          NL          0          0          1
      15 E3          NL          0          0          1
      16 e3          NL          0          0          1
      17 E4          NL          0          0          1
      18 e4          NL          0          0          1
      19 E5          NL          0          0          1
      20 e5          NL          0          0          1
      21 E6          NL          0          0          1
      22 e6          NL          0          0          1
      23 E7          NL          0          0          1
      24 e7          NL          0          0          1
      25 E8          NL          0          0          1
      26 e8          NL          0          0          1
      27 E9          NL          0          0          1
      28 e9          NL          0          0          1

Karush Kuhn Tucker optimality conditions:

KKT.PE: max.abs.err = 8.88e 16 on row 3
        max.rel.err = 8.91e 17 on row 3
        High quality

KKT.PB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

KKT.DE: max.abs.err = 1.89e 15 on column 10
        max.rel.err = 1.89e 15 on column 10
        High quality

KKT.DB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

End of output
```

B.3 Código do modelo por programação linear para a dureza

```
var x0;
var x1;
var x2;
var x3;
var x4;
var x5;
var x6;
var x7;
var x8;
var x9;

var E1>=0;
var e1>=0;
var E2>=0;
var e2>=0;
```

```
var E3>=0;
var e3>=0;
var E4>=0;
var e4>=0;
var E5>=0;
var e5>=0;
var E6>=0;
var e6>=0;
var E7>=0;
var e7>=0;
var E8>=0;
var e8>=0;
var E9>=0;
var e9>=0;
var E10>=0;
var e10>=0;

minimize z:E1+e1+E2+e2+E3+e3+E4+e4+E5+e5+E6+e6+E7+e7+E8+e8+E9+e9+E10+e10;
/*minimize z:x01+x02+x11+x12+x21+x22 +x31+x32 +x41+x42+x51+x52+x61+x62+x71+x72+x81+x82+x91+x92;*/



s.t. r1:x0 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9+E1 e1=1.252;
s.t. r2:x0+x1+1.5*x2+1.5*x3+1.5*x4+1.5*x5+2.25*x6+x7+2.25*x8+2.25*x9 +E2 e2=1.563;
s.t. r3:x0 + x1 + 2*x2 + 2*x3 + 2*x4 + 2*x5 + 4*x6 + x7 + 4*x8 + 4*x9+E3 e3=1.145;
s.t. r4:x0+1.5*x1+x2+1.5*x3+1.5*x4+2.25*x5+1.5*x6+2.25*x7+x8+2.25*x9+E4 e4=1.368;
s.t. r5:x0+1.5*x1+1.5*x2+2*x3+2.25*x4+3*x5+3*x6+2.25*x7+2.25*x8+4*x9+E5 e5=1.427;
s.t. r6:x0 + 1.5*x1 + 2*x2 + x3 + 3*x4 + 1.5*x5 + 2*x6 + 2.25*x7 + 4*x8 + x9+E6 e6=1.000;
s.t. r7:x0 + 2*x1 + x2 + 2*x3 + 2*x4 + 4*x5 + 2*x6 + 4*x7 + x8 + 4*x9+E7 e7=2.000;
s.t. r8:x0 + 2*x1 + 1.5*x2 + x3 + 3*x4 + 2*x5 + 1.5*x6 + 4*x7 + 2.25*x8 + x9 +E8 e8=1.631;
s.t. r9:x0 + 2*x1 + 2*x2 + 1.5*x3 + 4*x4 + 3*x5 + 3*x6 + 4*x7 + 4*x8 + 2.25*x9+E9 e9=1.310;
/*s.t. r10:E1 e1+E2 e2+E3 e3+E4 e4+E5 e5+E6 e6+E7 e7+E8 e8+E9 e9<=0.44;*/
/*s.t. r11:x0+x1+x2+x3+x4+x5+x6+x7+x8+x9=1.09;*/
s.t. r12:0*x0+0*x1+0*x2+0*x3+0*x4+0*x5+0*x6+0*x7+0*x8+0*x9+E10 e10=0;

 end;
```

B.4 Resultado da execução no Gusek do modelo de programação linear para o cálculo dos coeficientes para o modelo da dureza

```
Non zeros:   130
Status:      OPTIMAL
Objective:   z = 0 (MINimum)

   No.   Row name   St   Activity      Lower bound   Upper bound    Marginal

     1 z            B             0
     2 r1           NS        1.252         1.252             =       < eps
     3 r2           NS        1.563         1.563             =       < eps
     4 r3           NS        1.145         1.145             =       < eps
     5 r4           NS        1.368         1.368             =       < eps
     6 r5           NS        1.427         1.427             =       < eps
     7 r6           NS            1             1             =       < eps
     8 r7           NS            2             2             =       < eps
```

```
      9 r8              NS          1.631          1.631             =          < eps
     10 r9              NS           1.31           1.31             =          < eps
     11 r12             NS              0              0             =              1

    No. Column name    St    Activity      Lower bound    Upper bound    Marginal

      1 x0             B       0.733381
      2 x1             B        1.58267
      3 x2             B        4.47367
      4 x3             NF             0                                    < eps
      5 x4             B       0.995238
      6 x5             B       0.788571
      7 x6             B       0.229905
      8 x7             B       0.759048
      9 x8             B        1.17229
     10 x9             B       0.515619
     11 E1             NL             0              0                      1
     12 e1             NL             0              0                      1
     13 E2             NL             0              0                      1
     14 e2             NL             0              0                      1
     15 E3             NL             0              0                      1
     16 e3             NL             0              0                      1
     17 E4             NL             0              0                      1
     18 e4             NL             0              0                      1
     19 E5             NL             0              0                      1
     20 e5             NL             0              0                      1
     21 E6             NL             0              0                      1
     22 e6             NL             0              0                      1
     23 E7             NL             0              0                      1
     24 e7             NL             0              0                      1
     25 E8             NL             0              0                      1
     26 e8             NL             0              0                      1
     27 E9             NL             0              0                      1
     28 e9             NL             0              0                      1
     29 E10            NL             0              0                      2
     30 e10            B              0              0

Karush Kuhn Tucker optimality conditions:

KKT.PE: max.abs.err = 1.78e 15 on row 5
        max.rel.err = 1.01e 16 on row 5
        High quality

KKT.PB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

KKT.DE: max.abs.err = 2.83e 15 on column 9
        max.rel.err = 2.83e 15 on column 9
        High quality

KKT.DB: max.abs.err = 5.55e 17 on column 4
        max.rel.err = 5.55e 17 on column 4
        High quality

End of output
```

Apêndice C

Desempenho dos modelos por programação linear

Calculo dos coeficientes de desempenho do modelo HV usando o software Wolfram Mathematica

$yob = \{1.252, 1.563, 1.145, 1.368, 1.427, 1.000, 2.000, 1.631, 1.310\}$

$\text{FHV}(\text{x1\_}, \text{x2\_}, \text{x3\_}) = 0.759048\text{x1}^2 - 0.995238\text{x1x2} + 0.788571\text{x1x3} - 1.58267\text{x1} - 1.17229\text{x2}^2 + 0.229905\text{x2x3} + 4.47367\text{x2} - 0.515619\text{x3}^2 + 0\text{x3} - 0.733381$

$\text{yest} = \{\text{FHV}(1,1,1), \text{FHV}(1,1.5,1.5), \text{FHV}(1,2,2), \text{FHV}(1.5,1,1.5), \text{FHV}(1.5,1.5,2), \text{FHV}(1.5,2,1), \text{FHV}(2,1,2), \text{FHV}(2,1.5,1), \text{FHV}(2,2,1.5)\}$

$\text{MAPE} = \frac{1}{9}100 \left(\sum_{i=1}^{9} \frac{\text{yob}_{[[i]]} - \text{yest}_{[[i]]}}{\text{yob}_{[[i]]}} \right)$

$A = \frac{1}{9}100 \left(\sum_{i=1}^{9} \left(1 - \frac{\text{yob}_{[[i]]} - \text{yest}_{[[i]]}}{\text{yob}_{[[i]]}} \right) \right)$

$\text{RMSE} = \sqrt{\frac{1}{9} \sum_{i=1}^{9} (\text{yob}[[i]] - \text{yest}[[i]])^2}$

$U = \frac{\sqrt{\frac{1}{9} \sum_{i=1}^{9} (\text{yob}_{[[i]]} - \text{yest}_{[[i]]})^2}}{\sqrt{\frac{1}{9} \sum_{i=1}^{9} \text{yest}_{[[i]]}^2} + \sqrt{\frac{1}{9} \sum_{i=1}^{9} \text{yob}_{[[i]]}^2}}$

$\text{MSE} = \frac{1}{9} \sum_{i=1}^{9} (\text{yob}[[i]] - \text{yest}[[i]])^2$

$\text{mob} = \frac{1}{9} \sum_{i=1}^{9} \text{yob}[[i]]$

$\text{mest} = \frac{1}{9} \sum_{i=1}^{9} \text{yest}[[i]]$

$\text{R2} = \frac{\left(\sum_{i=1}^{9} (\text{yest}_{[[i]]} - \text{mest})(\text{yob}_{[[i]]} - \text{mob}) \right)^2}{\sum_{i=1}^{9} (\text{yob}_{[[i]]} - \text{mob})^2 \sum_{i=1}^{9} (\text{yest}_{[[i]]} - \text{mest})^2}$

Calculo dos coeficiêntes de AD usando o software wolfram mathematica.
$\text{yob} = \{1.09, 1.261, 1.562, 1.529, 1.431, 1., 2., 1.623, 1.755\}$

$\text{FAD}(\text{x1\_}, \text{x2\_}, \text{x3\_}) = 0.520381\text{x1}^2 - 0.767238\text{x1x2} + 0.183238\text{x1x3} + 0\text{x1} + 0.112381\text{x2}^2 + 0.787238\text{x2x3} - 0.422\text{x2} - 0.639619\text{x3}^2 + 0.698\text{x3} + 0.617619$

$\text{yest} = \{\text{FAD}(1,1,1), \text{FAD}(1,1.5,1.5), \text{FAD}(1,2,2), \text{FAD}(1.5,1,1.5), \text{FAD}(1.5,1.5,2), \text{FAD}(1.5,2,1), \text{FAD}(2,1,2), \text{FAD}(2,1.5,1), \text{FAD}(2,2,1.5)\}$

$\text{MAPE} = \frac{1}{9}100 \left(\sum_{i=1}^{9} \frac{\text{yob}_{[[i]]} - \text{yest}_{[[i]]}}{\text{yob}_{[[i]]}} \right)$

$A = \frac{1}{9}100 \left(\sum_{i=1}^{9} \left(1 - \frac{\text{yob}_{[[i]]} - \text{yest}_{[[i]]}}{\text{yob}_{[[i]]}} \right) \right)$

$\text{RMSE} = \sqrt{\frac{1}{9} \sum_{i=1}^{9} (\text{yob}[[i]] - \text{yest}[[i]])^2}$

$U = \frac{\sqrt{\frac{1}{9} \sum_{i=1}^{9} (\text{yob}_{[[i]]} - \text{yest}_{[[i]]})^2}}{\sqrt{\frac{1}{9} \sum_{i=1}^{9} \text{yest}_{[[i]]}^2} + \sqrt{\frac{1}{9} \sum_{i=1}^{9} \text{yob}_{[[i]]}^2}}$

$\text{MSE} = \frac{1}{9} \sum_{i=1}^{9} (\text{yob}[[i]] - \text{yest}[[i]])^2$

$\text{mob} = \frac{1}{9} \sum_{i=1}^{9} \text{yob}[[i]]$

$\text{mest} = \frac{1}{9} \sum_{i=1}^{9} \text{yest}[[i]]$

$\text{R2} = \frac{\left(\sum_{i=1}^{9} (\text{yest}_{[[i]]} - \text{mest})(\text{yob}_{[[i]]} - \text{mob}) \right)^2}{\sum_{i=1}^{9} (\text{yob}_{[[i]]} - \text{mob})^2 \sum_{i=1}^{9} (\text{yest}_{[[i]]} - \text{mest})^2}$

Apêndice D

Desempenho dos modelos calculados por mínimos quadrados

D.1 Desempenho do modelo de HV

$yob = \{1.252, 1.563, 1.145, 1.368, 1.427, 1., 2., 1.631, 1.31, 0\}$

$\sum_{i=1}^{10} \text{yob}[[i]]$

$\text{FHV}(\text{x1\_}, \text{x2\_}, \text{x3\_}) = 1.0524\text{x1}^2 - 0.408533\text{x1x2} + 0.201867\text{x1x3} - 2.6094\text{x1} - 0.878933\text{x2}^2 - 0.3568\text{x2x3} + 3.44693\text{x2} - 0.222267\text{x3}^2 + 1.02673\text{x3}$

$\text{yest} = \{\text{FHV}(1,1,1), \text{FHV}(1,1.5,1.5), \text{FHV}(1,2,2), \text{FHV}(1.5,1,1.5), \text{FHV}(1.5,1.5,2), \text{FHV}(1.5,2,1), \text{FHV}(2,1,2), \text{FHV}(2,1.5,1), \text{FHV}(2,2,1.5), \text{FHV}(0,0,0)\}$

$\text{MAPE} = \frac{1}{9}100 \left(\sum_{i=1}^{9} \frac{\text{yob}[[i]] - \text{yest}[[i]]}{\text{yob}[[i]]} \right)$

$A = \frac{1}{9}100 \left(\sum_{i=1}^{9} \left(1 - \frac{\text{yob}[[i]] - \text{yest}[[i]]}{\text{yob}[[i]]} \right) \right)$

$\text{RMSE} = \sqrt{\frac{1}{10} \sum_{i=1}^{10} (\text{yob}[[i]] - \text{yest}[[i]])^2}$

$U = \frac{\sqrt{\frac{1}{10} \sum_{i=1}^{10} (\text{yob}[[i]] - \text{yest}[[i]])^2}}{\sqrt{\frac{1}{10} \sum_{i=1}^{10} \text{yest}[[i]]^2} + \sqrt{\frac{1}{10} \sum_{i=1}^{10} \text{yob}[[i]]^2}}$

$k = 10$

$\text{MSE} = \frac{\sum_{i=1}^{k} (\text{yob}[[i]] - \text{yest}[[i]])^2}{k}$

$$\text{Eab} = \sum_{i=1}^{k} (\text{yob}[[i]] - \text{yest}[[i]])^2$$

$$\text{mob} = \frac{\sum_{i=1}^{k} \text{yob}_{[[i]]}}{k}$$

$$\text{mest} = \frac{\sum_{i=1}^{k} \text{yest}_{[[i]]}}{k};$$

$$\text{R2} = \frac{\left(\sum_{i=1}^{k} (\text{yest}_{[[i]]} - \text{mest})(\text{yob}_{[[i]]} - \text{mob})\right)^2}{\sum_{i=1}^{k} (\text{yob}_{[[i]]} - \text{mob})^2 \sum_{i=1}^{k} (\text{yest}_{[[i]]} - \text{mest})^2}$$

Apêndice E

Código principal da solução computacional

Aqui está o código de toda a solução computacional, conforme a Figura 4.1, o leitor poderá verificar a função main E.4 para ter uma ideia das rotinas que diretamente implementam o algorítimo MOPSO, *atualiza_velocidades*, *inicializa_particulas*, *e_dominada*, *atualiza_repositorio*, *analisa_dominancia_patual_pbest*. O leitor para ter uma ideia de como é a estrutura de dados de uma partícula dentro do código foi implementada basta verificar a seção E.2, dentro do código partícula é algo que muda de posição, ou seja, uma estrutura de dados que armazena posições, valores de funções em memória.

Código E.1: Rotinas

```
1  *****Arquivo PSO.c**********
2
3  #include <stdio.h>
4  #include <math.h>
5  #include <stdlib.h>
6  #include <time.h>
7  #include <string.h>
8
9  extern double (*HV)(double,double,double);
10 extern double (*AD)(double,double,double);
11
12 FILE* f;
13 FILE* f1;
14 FILE* f3;
15 FILE* f4;
16 extern int rand(void);
17 int  t;
18 int const  npsos=9;
19 int N=95;
20 int G=500;
21 double c1=1;
22 double c2=1;
23 double const x_min=1,x_max=2,y_min=1,y_max=2,z_min=1,z_max=2;
```

Código E.2: Particula

```
1  typedef struct{
2  double v[3];
3  /*velocidade nas 3 dimensoes*/
4  double p[3];
5  /*posicao nas 3 dimensoes*/
6  double vfp[2];
7  /*valores das funcoes objetivo na posicao p*/
8  double pbest[3];
9  /*a melhor posicao onde a particula esteve,com relacao a dominancia*/
```

```
10  double vfpbest[2];
11  /*valores das funcoes objetivo em pbest,ex:vfpbest[0]=fo1(pbest[0],pbest[1],pbest[2]);
12  vfpbest[1]=fo2(pbest[0],pbest[1],pbest[2]);*/
13  double gbest[3];
14  /*vai ser usado no calculo da velocidade e pego do repositorio, a posicao */
15  int usad;
16  }pso;
```

Código E.3: Rotinas

```
1   pso particulas[npsos];
2
3   int k;
4   double w(int i)
5   {
6
7     return ( (0.9 0.4)*((G i)/G) + 0.4);
8   }
9
10
11  int atualiza_repositorio(pso* ,pso* r);
12
13  int e_dominada(int i ,pso r[]);
14
15  void atualiza_velocidades(pso r[],int i);
16
17
18  void inicializa_particulas(pso r[],pso A[])
19  {
20
21
22
23    for(int j=0 ;j < npsos ; j++){
24
25     /*Inicializacao padrao do algoritimo de particulas de 1995*/
26     /*x_max,y_max,z_max,..sao os limites do espaco de busca*/
27     r[j].p[0]=x_min+((double)rand()/(RAND_MAX))*(x_max x_min);
28     r[j].p[1]=y_min+((double)rand()/(RAND_MAX))*(y_max y_min);
29     r[j].p[2]=z_min+((double)rand()/(RAND_MAX))*(z_max z_min);
30
31     r[j].v[0]=x_min+((double)rand()/(RAND_MAX))*(x_max x_min);
32     r[j].v[1]=y_min+((double)rand()/(RAND_MAX))*(y_max y_min);
33     r[j].v[2]=z_min+((double)rand()/(RAND_MAX))*(z_max z_min);
34
35
36
37     r[j].pbest[0]=x_min+((double)rand()/(RAND_MAX))*(x_max x_min);
38     r[j].pbest[1]=y_min+((double)rand()/(RAND_MAX))*(y_max y_min);
39     r[j].pbest[2]=z_min+((double)rand()/(RAND_MAX))*(z_max z_min);
40
41     r[j].gbest[0]=x_min+((double)rand()/(RAND_MAX))*(x_max x_min);
42     r[j].gbest[1]=y_min+((double)rand()/(RAND_MAX))*(y_max y_min);
43     r[j].gbest[2]=z_min+((double)rand()/(RAND_MAX))*(z_max z_min);
44
45      r[j].vfp[0]=HV(r[j].p[0], r[j].p[1], r[j].p[2]);
46      r[j].vfp[1]=AD(r[j].p[0], r[j].p[1], r[j].p[2]);
47
48      r[j].vfpbest[0]=HV(r[j].pbest[0], r[j].pbest[1], r[j].pbest[2]);
49      r[j].vfpbest[1]=AD(r[j].pbest[0], r[j].pbest[1], r[j].pbest[2]);
50
51     /* printf("x=%f,y=%f,z=%f,f1=%f,f2=%f \n",r[j].p[0],r[j].p[1],r[j].p[2],r[j].vfp[0],r[j].vfp[1]);*/
52
53       fprintf(f,"{␣%f,␣%f,␣%f}␣,␣", r[j].p[0], r[j].p[1], r[j].p[2]);
54       fprintf(f1,"{␣%f,␣%f}␣,␣", r[j].vfp[1], r[j].vfp[0]);
55
56
57
58    }
59
60
61   for(int j=0;j<npsos;j++)
62  {    /* printf("%d \n",j);*/
63       if(e_dominada(j,r)==1){
64             /* printf("x particula nao dominada %d , f1=%f , f2=%f \n ",j,r[j].vfp[0],r[j].vfp[1]);*/
65             /* printf("{%f,%f} ,",r[j].vfp[0],r[j].vfp[1]);*/
66              int local=atualiza_repositorio(&r[j],A);
67              printf("atualizou␣posicao=%d␣repositorio␣%f\n", local,A[0].vfp[0]);
68         }
69  }
```

```
70
71
72
73  }
74
75  int e_dominada(int i,pso r[])
76  {
77   int nao_dominada=1;
78   for(int j=0 ;j < npsos ; j++)
79   {   if(j==i) continue;
80     /* A particula 1 nao pode ser pior que a particula
81  2 em nenhum objetivo,se for pior nao e uma solucao nao dominada*/
82       if( (r[i].vfp[0]<r[j].vfp[0] && r[i].vfp[1]<r[j].vfp[1])
83       ||(r[i].vfp[0]<r[j].vfp[0] && r[i].vfp[1]==r[j].vfp[1])
84       ||(r[i].vfp[0]==r[j].vfp[0] && r[i].vfp[1]<r[j].vfp[1]))
85        {
86         nao_dominada=0;
87         break;
88
89        }
90        /*a particula nao e pior que a particula 2 em nenhum objetivo,
91    resta verificar se e estritamente melhor em pelo menos 1 objetivo*/
92     /* if (r[i].vfp[0]>r[j].vfp[0] || r[i].vfp[1]>r[j].vfp[1]){
93          nao_dominada=1;
94       } */
95   }
96
97    return nao_dominada;
98  }
99
100
101 int atualiza_repositorio(pso* q,pso* r)
102 {
103 static int rx=0;
104  /*if(signal(SIGSEGV,handler)==SIG_ERR)
105     printf("%s\n","sinal");
106   signal(SIGSEGV,SIG_IGN);*/
107   pso p = *q;
108   p.usad=0;
109   int dominancia=1;
110   printf("␣testando␣atualiza␣repositorio␣p.vfp[0]=%f␣",p.vfp[0]);
111   int pos_in=1000;
112     for(int j=0 ;j < N ; j++)
113     {
114
115       printf("r[%d].usad=%d\n",j,r[j].usad);
116     if((r[j].usad==0) || (r[j].usad==1))
117      {pos_in=(int)fmin(j,pos_in);}
118      else{printf("%s,r[%d].usad=%d\n","repositorio␣cheio\n",j,r[j].usad);/*return  1;*/}
119
120     if( (p.vfp[0]<r[j].vfp[0] && p.vfp[1]<r[j].vfp[1])
121     || (p.vfp[0]<r[j].vfp[0] && p.vfp[1]==r[j].vfp[1])
122     ||(p.vfp[0]==r[j].vfp[0] && p.vfp[1]<r[j].vfp[1]))
123     {
124             dominancia=0;
125             break;
126     }
127
128
129     }
130   if(dominancia==0) return  1;
131  /*insere p*/
132   p.usad=1;/*insere no repositorio a particula com a flag usad=1*/
133   if(pos_in==1000)pos_in=rx++ % N;
134   r[pos_in]=p;
135   printf( "vai␣inserir␣a␣particula␣r[%d].p[0].p[1].p[2]=%f,%f,%f\n",
136   pos_in, r[pos_in].p[0],r[pos_in].p[1],r[pos_in].p[2]);
137   compacto[pos_in]=1;
138   for(int j=0 ;j < N ; j++)
139   {
140
141     /*Retirando as particulas do repositorio final que p domina*/
142      if ((r[j].vfp[0]<p.vfp[0] && r[j].vfp[1]<p.vfp[1])
143      ||(r[j].vfp[0]<p.vfp[0] && r[j].vfp[1]==p.vfp[1])
144      ||(r[j].vfp[0]==p.vfp[0] && r[j].vfp[1]<p.vfp[1]))
145        {r[j].usad= 1;
146        compacto[j]=0;
147        printf("retirando␣r[%d]\n",j);
148        }
149
```

```
150       /*if ((r[j].vfp[0]>p.vfp[0] || r[j].vfp[1]>p.vfp[1])==0){r[j].usad=1;}*/
151
152    }
153
154    printf("inserido na posicao %d repositorio\n",pos_in);
155    return pos_in;
156   }
157
158   int analisa_dominancia_patual_pbest(pso p)
159   {
160      /* A particula 1 nao pode ser pior que a particula
161   2 em nenhum objetivo,se for pior nao e uma solucao nao dominada*/
162       if (p.vfp[0]<p.vfpbest[0] || p.vfp[1]<p.vfpbest[1]) return 0;
163     /*Se nao foi reprovada no teste anterior a particula nao e pior que a particula 2
164     em nenhum objetivo, resta verificar
165     se e estritamente melhor em pelo menos 1 objetivo*/
166       if (p.vfp[0]>p.vfpbest[0] || p.vfp[1]>p.vfpbest[1]) return 1;
167       return  1;
168   }
169
170
171
172
173   int analisa_dominancia_pbest_patual(pso p)
174   {
175         /* A particula 1 nao pode ser pior que a particula
176   2 em nenhum objetivo,se for pior nao e uma solucao nao dominada*/
177       if (p.vfpbest[0]<p.vfp[0] || p.vfpbest[1]<p.vfp[1]) return 0;
178     /*Se nao foi reprovada no teste anterior
179     a particula nao e pior que a particula 2
180     em nenhum objetivo,
181     resta verificar se e estritamente
182     melhor em pelo menos 1 objetivo*/
183       if (p.vfpbest[0]>p.vfp[0] || p.vfpbest[1]>p.vfp[1]) return 1;
184       return  1;
185   }
186
187   void atualiza_velocidades(pso r[],int i)
188   {
189
190     /*checa_nao_dominadas(repositorio,rcompacto,t);*/
191
192       for(int j=0 ;j < npsos ; j++)
193     {  /*as particulas devem ficar dentro dos limites do espaco de busca*/
194
195        /*select lider*/
196       /* selecionar_lider(r,rcompacto,t);*/
197
198
199       /* gx=x_min+((double)rand()/(RAND_MAX))*(x_max x_min);
200        gy=y_min+((double)rand()/(RAND_MAX))*(y_max y_min);
201        gz=z_min+((double)rand()/(RAND_MAX))*(z_max z_min);*/
202
203
204         r[j].gbest[0]=x_min+((double)rand()/(RAND_MAX))*(x_max x_min);
205         r[j].gbest[1]=y_min+((double)rand()/(RAND_MAX))*(y_max y_min);
206         r[j].gbest[2]=z_min+((double)rand()/(RAND_MAX))*(z_max z_min);
207
208    /* r[j].gbest[0]= fmax(x_min,fmin(x_max,r[j].p[0]+
209    ((double)rand()/(RAND_MAX))*( 1.74473 + 2.65147*r[j].p[0]
210    1.66987*r[j].p[1] +
211    0.8792*r[j].p[2])));
212    r[j].gbest[1]=fmax(x_min,fmin(x_max,r[j].p[1]+
213    ((double)rand()/(RAND_MAX))*
214    (3.8896    1.66987*r[j].p[0]    2.0272*r[j].p[1] +
215    0.924533*r[j].p[2])));
216    r[j].gbest[2]=fmax(x_min,fmin(x_max,r[j].p[2]+
217   ((double)rand()/(RAND_MAX))*(0.860067 + 0.8792*r[j].p[0] +
218   0.924533*r[j].p[1]    2.21787*r[j].p[2])));*/
219
220
221
222         r[j].vfpbest[0]=HV( r[j].pbest[0], r[j].pbest[1], r[j].pbest[2]);
223         r[j].vfpbest[1]=AD( r[j].pbest[0], r[j].pbest[1], r[j].pbest[2]);
224
225        r[j].v[0]=(w(i)*r[j].v[0] +c1*((double)rand()/RAND_MAX )*
226        (r[j].pbest[0] r[j].p[0]) + c2*((double)rand() /RAND_MAX)*(r[j].gbest[0]  r[j].p[0]));
227        r[j].v[1]=(w(i)*r[j].v[1] +c1*((double)rand()/RAND_MAX )*
228        (r[j].pbest[1] r[j].p[1]) + c2*((double)rand() /RAND_MAX)*(r[j].gbest[1]    r[j].p[1]));
229        r[j].v[2]=(w(i)*r[j].v[2] +c1*
```

```
230         ((double)rand() /RAND_MAX)*(r[j].pbest[2] r[j].p[2]) +
231         c2*((double)rand() /RAND_MAX )*(r[j].gbest[2] r[j].p[2]));
232
233
234
235       /* r[j].v[0]=fmax(x_min,fmin(x_max,w(i)*r[j].v[0] +
236       c1*((double)rand()/RAND_MAX )*(r[j].pbest[0] r[j].p[0]) +
237       c2*((double)rand() /RAND_MAX)*(r[j].gbest[0] r[j].p[0])));
238        r[j].v[1]=fmax(y_min,fmin(y_max,w(i)*r[j].v[1] +
239        c1*((double)rand()/RAND_MAX )*(r[j].pbest[1] r[j].p[1]) +
240        c2*((double)rand() /RAND_MAX)*(r[j].gbest[1] r[j].p[1])));
241        r[j].v[2]=fmax(z_min,fmin(z_max,w(i)*r[j].v[2] +
242        c1*((double)rand() /RAND_MAX)*(r[j].pbest[2] r[j].p[2]) +
243        c2*((double)rand() /RAND_MAX )*(r[j].gbest[2] r[j].p[2])));*/
244
245       /* printf("particula %d pbestx=%f pbesty=%f pbestz= %f\n", j,r[j].pbest[0], r[j].pbest[1], r[j].pbest[2]);*/
246
247
248
249       /* r[j].p[0]= fmax(x_min,fmin(x_max ((double)rand()/(RAND_MAX))*(0.00001),r[j].p[0] + r[j].v[0]));*/
250        r[j].p[0]= fmax(x_min,fmin(x_max,r[j].p[0] + r[j].v[0]));
251        r[j].p[1]= fmax(y_min,fmin(y_max,r[j].p[1] + r[j].v[1]));
252        r[j].p[2]= fmax(z_min,fmin(z_max,r[j].p[2] + r[j].v[2]));
253
254
255
256       r[j].vfp[0]=HV( r[j].p[0], r[j].p[1], r[j].p[2]);
257       r[j].vfp[1]=AD( r[j].p[0], r[j].p[1], r[j].p[2]);
258
259       /*  printf("{ %f, %f, %f} , ", r[j].p[0], r[j].p[1], r[j].p[2]);*/
260
261
262         fprintf(f,"{ %f, %f, %f} , ", r[j].p[0], r[j].p[1], r[j].p[2]);
263         fprintf(f1,"{ %f, %f} , ", r[j].vfp[1], r[j].vfp[0]);
264
265
266
267          for(int j=0;j<npsos;j++)
268            {   /* printf("%d \n",j);*/
269              if(e_dominada(j,r)==1){
270
271                 fprintf(f4,"{ %f, %f} , ", r[j].vfp[1], r[j].vfp[0]);
272
273                 }
274           }
275
276
277
278        if(analisa_dominancia_patual_pbest(r[j])==1)
279        {
280            r[j].pbest[0]=r[j].p[0];
281            r[j].pbest[1]=r[j].p[1];
282            r[j].pbest[2]=r[j].p[2];
283
284            r[j].vfpbest[0]=HV( r[j].pbest[0], r[j].pbest[1], r[j].pbest[2]);
285            r[j].vfpbest[1]=AD( r[j].pbest[0], r[j].pbest[1], r[j].pbest[2]);
286
287
288         }
289
290
291
292       /* printf("vx=%f vy=%f vz= %f\n", r[j].v[0], r[j].v[1], r[j].v[2]);
293
294
295        printf("x=%f y=%fz= %f\n", r[j].p[0], r[j].p[1], r[j].p[2]);*/
296
297
298     }
299   }
```

Código E.4: Rotina main

```
1   int main(){
2   srand (time(NULL));
3   t=5;
4
5   for(int k=0;k<N;k++)
6   {
```

```
   repositorio[k].usad=0;
}

f = fopen("particula.txt", "w");
f1 = fopen("f1xf2.txt", "w");
f3=fopen("pareto.txt","w");
f4=fopen("naodominada.txt","w");

 fprintf(f,"%s","ListPointPlot3D[{");
 fprintf(f1,"%s","ListPlot[{");

FILE* f5;
f5 = fopen("latex.txt", "w");

inicializa_particulas(particulas,repositorio);

for(int i=0;i<G;i++){
atualiza_velocidades(particulas,i);
/*For das particulas*/
   for(int j=0;j<npsos;j++)
   {
     if(e_dominada(j,particulas)==1){

            int local=atualiza_repositorio(&particulas[j],repositorio);

      }
   }


   }

 for(int k=0;k<N;k++)
 {
   if(repositorio[k].usad==1)

      fprintf(f3,"{ %f, %f} , ", repositorio[k].vfp[1], repositorio[k].vfp[0]);

 }


 fprintf(f,"%s", "}]");
 fprintf(f1,"%s", "}]");


for(int k=0;k<N;k++)

 if(repositorio[k].usad==1)

fprintf(f5," %d , %f , %f , %f,%f,%f \n",k,
repositorio[k].vfp[1],
repositorio[k].vfp[0],
repositorio[k].p[0],
repositorio[k].p[1],
repositorio[k].p[2]);

 }
fclose(f);
fclose(f1);
fclose(f3);
fclose(f4);
fclose(f5);
return 0;
}
```

Código E.5: Modelo QM

```
****Arquivo modqum.c*****

double (*HV)(double,double,double);
double (*AD)(double,double,double);
double fo1(double x,double y,double z)
{/*HV*/

  return (2.609400*x + 3.446933*y +
  1.026733*z    0.408533*x*y +
  0.201866*x*z    0.356800*y*z +
  1.052400*x*x    0.878933*y*y
  0.222266*z*z);

```

```
14  }
15  double fo2(double x,double y,double z)
16  {/*AD*/
17    return (0.864666*x + 0.442666*y
18    0.166666*z    1.261333*x*y +
19    0.677333*x*z + 1.281333*y*z +
20    0.273333*x*x    0.134666*y*y
21    0.886666*z*z);
22  }
23
24
25
26  void link(){
27  HV=fo1;
28  AD=fo2;
29
30  }
31  void __attribute__((constructor)) inicializa_load(void)
32  {
33    link();
34  }
```

Código E.6: Modelo PL

```
1   ****Arquivo mod.c******
2
3   double (*HV)(double,double,double);
4   double (*AD)(double,double,double);
5   double fo1(double x,double y,double z)
6   {/*HV*/
7     return ( 0.733381    1.58267*x  + 4.47367*y + 0*z    0.995238*x*y +
8    0.788571*x*z + 0.229905*y*z + 0.759048*x*x    1.17229*y*y
9    0.515619*z*z);
10  }
11  double fo2(double x,double y,double z)
12  {/*AD*/
13    return (0.617619 + 0*x    0.422*y + 0.698*z    0.767238*x*y +
14   0.183238*x*z + 0.787238*y*z + 0.520381*x*x + 0.112381*y*y
15   0.639619*z*z);
16  }
17
18
19
20  void link(){
21  HV=fo1;
22  AD=fo2;
23
24  }
25  void __attribute__((constructor)) inicializa_load(void)
26  {
27    link();
28  }
```

Código E.7: Modelo do Artigo

```
1   **** Arquivo moda.c ******
2
3   double (*HV)(double,double,double);
4   double (*AD)(double,double,double);
5   double fo1(double x,double y,double z)
6   {/*HV*/
7
8     return (0.583    0.930*x + 2.076*y    0.258*z + 0.422*x*x+ 0.798*y*y +
9     0.156*z*z);
10
11  }
12  double fo2(double x,double y,double z)
13  {/*AD*/
14
15   return (0.875    2.136*x    2.771*y + 5.022*z + 1.140*x*x + 0.740*y*y
16    0.337*z*z+ 0.787*x*y    1.519*x*z    0.771*y*z);
17
18  }
19
20
21
22  void link(){
23  HV=fo1;
```

```
24  AD=fo2;
25
26  }
27  void __attribute__((constructor)) inicializa_load(void)
28  {
29    link();
30  }
```